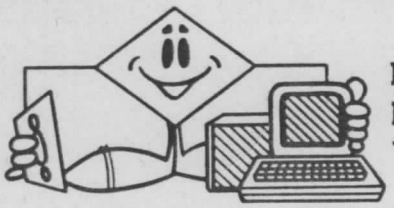


HARDWARE

Costruiamo una FORTH-Card con il chip RSC-65F11

Uno degli integrati più attraenti apparsi recentemente sul mercato è il chip RSC-65F11, un single-chip basato sulla architettura della 6502, che contiene un completo sistema operativo FORTH. Con esso, aggiungendo una manciata di componenti esterni, si mette insieme un FORTH-Microcomputer. Questo, ed altri articoli che seguiranno, descrivono ampiamente le caratteristiche funzionali, nonché le modalità implementative, del sistema che abbiamo pensato di battezzare FORTH-Card. Esso sarà anche disponibile in forma di kit per tutti coloro che vorranno avvicinarsi in modo concreto al mondo affascinante del linguaggio FORTH ed alle sue notevoli potenzialità.

Parte prima

di **Paolo Bozzola**
Computerjob Elettronica - Brescia

Introduzione

Confesso che provo una strana malinconia al pensiero dei primi pionieristici esperimenti per i quali, sul nascere della mia passione per l'elettronica, reperivo in angusti negozietti i transistori dalle sigle più strane, con lo scopo di potenziare ed addolcire gli aspri, striduli suoni delle chitarre elettriche dei miei compagni di liceo: erano i tempi magici degli effetti speciali e delle prime applicazioni di "musica elettronica".
Ora, invece, il dio Soft imperversa e, se mi trovo seduto davanti al terminale per scrivere un articolo per **Bit**, lo sono perché praticamente costretto dal mio lavoro ad editare righe e righe di codice.
Che cosa c'entra questo, si dirà, con la FORTH-Card? Ebbene, un nesso esiste, nel senso che, incredibilmente, lavorando su questo progetto mi sono sentito catapultare alle origini, con l'ebbrezza antica provocata dallo sfrigolio dello stagno e dall'odore della pasta salda, condita dal brivido di suspence sulla scommessa se il marchingegno avrebbe funzionato una volta montato. Oltretutto, come resistere all'attrazione di un completo sistema di sviluppo che si può realizzare - udite! udite! - con soli cinque chip?
Detto e fatto, ho proprio voluto vedere se era vero, ed i risultati sono stati tanto positivi da indurmi non solo a proporre alla rivista una descrizione della mia esperienza, ma anche da invogliarmi a riprendere carta e matita per rifare il progetto, in modo che assumesse una forma definitiva, dunque meno spartana del frettoloso cablaggio iniziale.

Certo che il superchip ha ammaliato non pochi lettori (soprattutto se già fiutano quel che può fare il FORTH).
Inizio senza indugio la discussione, che conterrà richiami sulla struttura del linguaggio, oltre che diffuse descrizioni funzionali e tutorial sulla macchina che andremo a realizzare.

Concetti del FORTH

I lettori di **Bit** non sono certo impreparati sull'argomento FORTH: gli articoli di Marco Tausel e di Marco Morocutti, citati in bibliografia, sono abbastanza recenti. Tuttavia mi sembra giusto, per comodità di molti, riprendere a grandi linee l'argomento, anche perchè, con le idee chiare su questo linguaggio, si capirà meglio il funzionamento del sistema RSC-65F11.
Essendo stato in origine sviluppato proprio per controlli in tempo reale, il FORTH ha caratteristiche tali da renderlo ideale per il controllo di macchine e di processi, acquisizione di dati, gestione ambientale ed energetica, robotica, test automatici e simili. In molte di queste applicazioni, infatti, sarebbe necessaria la velocità del linguaggio macchina, mentre la complessità dei programmi obbligherebbe l'uso di un linguaggio ad alto livel-

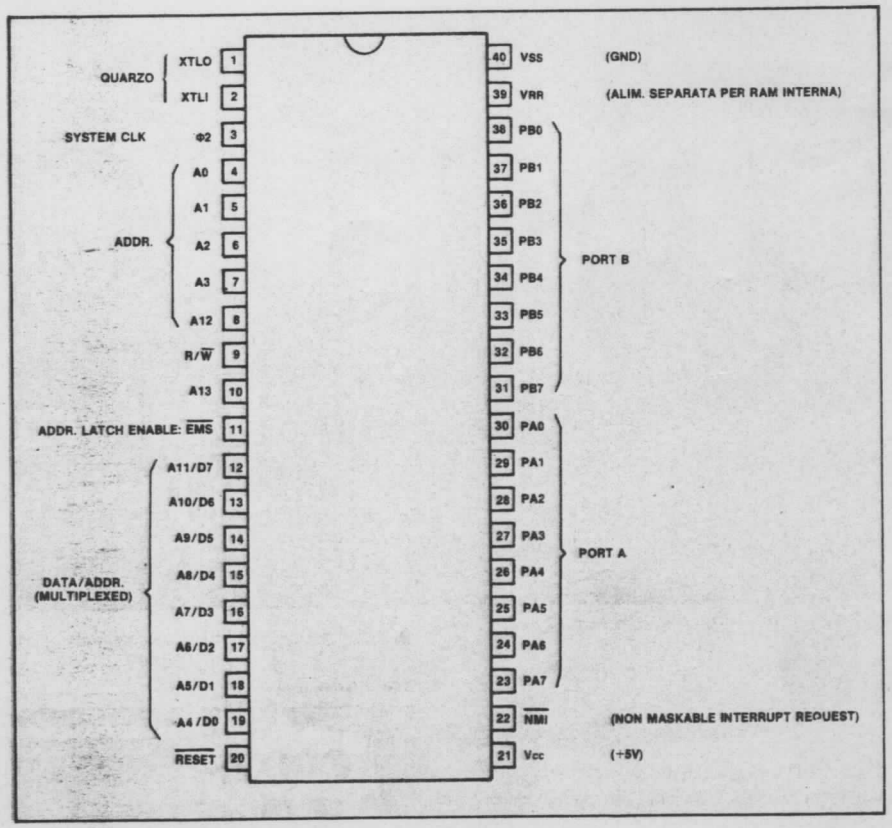


Figura 1 - Piedinatura dell'RSC-65F11.

lo, per facilitarne la stesura ed il debugging. Ebbene, il FORTH è stato progettato per soddisfare sia le esigenze di velocità che quelle di efficienza nella stesura dei programmi.

Oltre che di un linguaggio, si tratta, insieme, di un sistema operativo, un compilatore interattivo, una struttura di dati, o un interprete: dipende dal punto di vista dell'utente. Combinando la potenza dei compilatori con la versatilità degli interpreti, il risultato è unico: si danno operazioni predefinite che minimizzano il tempo ed i costi di sviluppo del software. Il FORTH supporta una programmazione strutturata ed altamente modulare, compila in modo interattivo - facilitando il debugging - e produce uno pseudocodice oggetto di ridottissime dimensioni, che viene eseguito con la massima velocità.

Tutte queste virtù sono infuse nell'RSC-65F11, del quale la figura 1 mostra la piedinatura, la figura 2 mostra la struttura interna e la figura 3 elenca le caratteristiche principali.

Il nostro single-chip, prodotto dalla Rockwell, contiene in pratica una CPU 6502 migliorata, 192 byte di RAM, 16 oppure 40 linee di I/O programmabili da software, due contatori/timer a 16 bit, un port seriale e dieci sorgenti di interrupt, di cui una esterna. Inoltre il chip, com'è da attendersi dopo quanto si è detto, contiene circa 3 Kbyte di ROM su cui risiede il nucleo (kernel) del sistema operativo FORTH. Infine vi sono le linee dei dati/indirizzi (multiplexate), che permettono di collegare all'esterno fino a 16 Kbyte di memoria. Incredibile, ma vero: il kernel dell'RSC-65F11 contiene anche tutte le routine di gestione di una memoria di massa su dischi da 5"1/4. Basta prevedere, sul bus, un comune chip floppy disk controller della Western Digital ed immediatamente si possono collegare fino a quattro drive, anche dei tipi a doppia testa e 96 T.P.I., dopodiché si lavora persino in memoria virtuale!

Tornando al FORTH occorre riconoscere che non è così semplice come può sembrare, se non altro per il fatto che è estremamente diverso, nella forma e nella filosofia, da linguaggi come, per esempio, il BASIC, il FORTRAN o il Pascal. Cercherò allora, magari senza essere troppo pedante, di dare una panoramica e delle caratteristiche e dell'implementazione del FORTH, con un costante riferimento a come tutto ciò è realizzato nell'RSC-65F11.

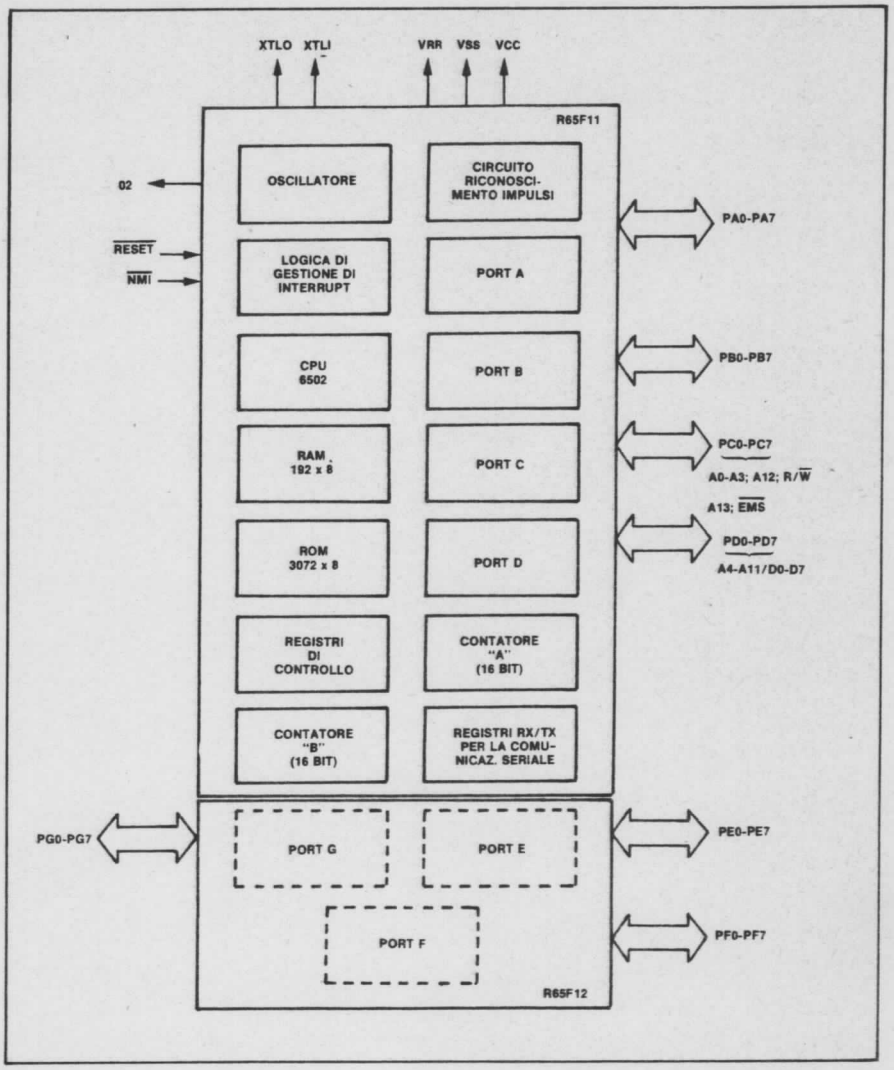


Figura 2 - Struttura a blocchi dell'RSC-65F11: la parte in basso è peculiare dell'RSC-65F12 (che ha tre port di I/O in più).

Caratteristiche del FORTH

Innanzitutto, il FORTH è estensibile, il che significa che si possono creare ed aggiungere nuove operazioni alla struttura base. Le operazioni, che in FORTH si chiamano WORD (parole), sono definite sfruttando word già esistenti e che a loro volta erano state definite con word ancor più primitive. Inoltre le nuove word possono anche essere definite in termini di linguaggio Assembly (nel caso, sarà il linguaggio macchina della 6502), e la creazione di sempre nuove word procede in modo talmente strutturato che, alla fine, tutto il programma sarà definito da un'unica word. Eseguendola si esegue il programma, il che si effettua di norma dalla consolle di sistema, battendone semplicemente il nome seguito da RETURN. Si può comunque organizzare il sistema in modo che, subito dopo l'accensione, la suddetta word sia subito eseguita, realizzando così un sistema munito di autostart. Se si tralascia per il momento il fatto che le nuove word possono risiedere in RAM (o in EPROM), per il resto non vi è alcuna distinzione fra word facenti parte del nucleo originario in ROM e le word create dal programmatore: il FORTH è quindi un linguaggio che può crescere su se stesso. Tale proprietà permette all'utente di definire diverse e separate librerie e addirittura di inventare nuovi nomi alle word esistenti, per adattarle meglio al problema che deve risolvere.

Il FORTH tiene tutte le word in un vocabolario, che include praticamente tutto il codice oggetto delle word del sistema e di quelle create dall'utente per la sua particolare applicazione. Le strutture di dati possono essere nel vocabolario o fuori di esso, secondo le necessità. La struttura interna del vocabolario è uniforme ed assai più semplice di altre strutture interne di altri linguaggi, per cui è tipica dei programmatori in FORTH una buona conoscenza non solo della sintassi, ma

- Kernel (nucleo) del FORTH in ROM.
- CPU: 6502 migliorata (4 istruzioni in più).
- RAM interna, statica, 192 byte.
- 16 linee di I/O bidirezionali, TTL-compatibili (RSC-65F11) oppure 40 linee (RSC-65F12).
- Un port a 8 bit con ingresso memorizzabile da latch.
- Due timer/counter programmabili a 16 bit, con latch, per i seguenti usi:
— misura della durata di impulsi;
— generazione di forme d'onda;
— generazione di impulsi;
— temporizzatori;
— contatori di eventi;
— temporizzatori retriggerabili.
- Port di comunicazione seriale:
— operazione in modo asincrono/full duplex;
— lunghezza di carattere da 5 a 8 bit;
— possibilità di "wake-up";
— modo shift-register sincrono;
— baud rate standard, programmabile, fino a 62500 baud.
- Dieci sorgenti di interrupt:
— quattro ingressi sensibili alle transizioni: due 1→0 e due 0→1;
— reset;
— interrupt non mascherabile (NMI);
— due interrupt dai due time/counter;
— interrupt da dato seriale ricevuto;
— interrupt da dato seriale trasmesso.
- Memoria esterna indirizzabile dino a 16 Kbyte.
- Circuito di clock versatile: 1 o 2 MHz, con quarzo applicato direttamente o ingresso da oscillatore esterno.
- 1μs minimo di tempo di istruzione a 2 MHz.
- Tecnologia NMOS silicon gate depletion load.
- Singola alimentazione a +5V.
- 32 byte dalla RAM interna sono mantenibili in stand-by con batteria esterna di back-up (terminale V_{rr}) con consumo di 12 mA.
- Contenitore 40 piedini DIP (RSC-65F11).
- Contenitore 64 piedini QUIP (RSC-65F12).

Figura 3 - Tavola delle caratteristiche principali del chip RSC-65F11.

anche del meccanismo interno del linguaggio (il che è di solito tabù per la maggioranza dei programmatori BASIC).

Il FORTH compila le word definite dal programmatore, ed il codice oggetto così prodotto è estremamente compatto, persino più compatto dell'analogo ottenibile con l'Assembly in linguaggio macchina (ricordarsi comunque che è un codice intermedio, poi interpretato. Ndr). Nel FORTH dell'RSC-65F11 la compattezza è stata particolarmente curata; va poi tenuto conto che, siccome la maggior parte dei calcoli e delle operazioni è compito di word esistenti nel kernel (scritto sulla ROM interna), l'utente non deve ogni volta aggiungere alle sue routine il famoso "run-time package" necessario per altri linguaggi compilati. Per meglio dire, esso c'è, ma non si vede e non occorre preoccuparsene.

Conseguenze di questa impostazione sono un gran risparmio di memoria ed una relativamente piccola perdita nella velocità di esecuzione, cosa che però è largamente compensata dalla possibilità di trattare convenientemente (le word sono già pronte) numeri a 16 o più bit, con tutte le operazioni aritmetiche e logiche comunemente usate.

Il FORTH è un linguaggio altamente strutturato: l'istruzione GO TO o simili vi è bandita, per cui lo sviluppo dei programmi deve necessariamente seguire le regole di una programmazione strutturata, con lo sviluppo top-down e l'immissione ed il debug bottom-up. Come il più noto dei linguaggi strutturati, il Pascal, il FORTH ha strutture di controllo del tipo IF...ELSE, con i loop controllati da DO..UNTIL e WHILE. Durante la stesura di una word tali strutture possono essere concatenate senza limitazioni.

Il FORTH usa uno stack (catasta) come area di scambio dei parametri fra le varie subroutine, le quali altro non sono, in pratica, che le word. L'ordine con cui le varie operazioni sono eseguite corrisponde alla Notazione Polacca Inversa (RPN, Reverse Polish Notation), nella quale i simboli dell'operazione che si va a compiere seguono gli operandi. È un sistema ben noto, ad esempio, agli utenti HP, per cui una operazione che in BASIC è <2+2> in FORTH diventa <2 2+.

Per quali motivi il FORTH usa lo stack (ed obbliga ad usarlo) in modo così marcato, mentre tutti gli altri linguaggi lo nascondono e cercano di evitare che l'utente possa accedervi? Perché si usa la notazione polacca invece della più familiare forma diretta?

Parte della risposta discende dal fatto, assai importante, che l'implementazione di uno stack fa sì che vi sia pochissimo lavoro nella gestione dei collegamenti (linking) fra le varie subroutine chiamate in sequenza, che poi altro non sono che le word del FORTH. Il FORTH, grazie allo stack, riduce a zero il costo delle subroutine, tant'è vero che tutto il linguaggio è costruito su chiamate di subroutine. Grazie allo stack le word possono accettare e restituire un numero qualsiasi di argomenti, senza che il fatto comporti definizioni complicate di variabili locali, generali, statiche e via dicendo.

Inoltre l'uso dello stack incoraggia la stesura di programmi in forma modulare, il che significa poter verificare (operazione chiamata "debugging") i vari moduli separatamente e con la massima facilità. Consideriamo infatti l'ambiente FORTH: ogni modulo (cioè ogni word od operazione o procedura o subroutine che dir si voglia) ha uno ed un solo punto di entrata ed uno ed un solo punto di uscita, obbligatoriamente. Tutta la comunicazione con l'esterno, cioè l'acquisizione di dati e la memorizzazione dei risultati, avviene tramite lo stack, per cui non possono esservi effetti collaterali sulle altre word, variabili, ecc., a meno che ciò sia espressamente voluto, per qualche artificio, dal programmatore. E poi una word è un brevissimo pezzetto di codice, tipicamente di tre o quattro righe, per cui più piccolo è il modulo e più facile sarà l'operazione di test.

Il FORTH è un linguaggio interattivo. Il test di ogni word è quindi immediato, perché praticamente tutte le word del FORTH possono essere eseguite direttamente come comandi inviati dalla consolle (terminale video) collegata con una linea seriale al sistema di sviluppo. Non solo, ma eseguite così le word si comportano in ogni caso allo stesso modo che se fossero eseguite da programma (cioè chiamate da altre word del programma). Per un semplice test l'utente può pre-caricare lo stack con dati particolari, eseguire la word e poi esaminare lo stack per controllare che i risultati ivi lasciati siano come egli si aspettava.

Non solo, ma la verifica dei moduli in FORTH raramente richiede l'esame di un qualche codice, a parte il singolo nome della word sotto test: infatti, l'ottica del

FORTH è quella di avere un "glossario" di definizioni, ove sono descritte in forma sintetica le word, dichiarando i parametri in ingresso, quelli lasciati in uscita, assieme ad una breve storia della word stessa. Il programmatore così non ha alcuna necessità di scorrere un listato di codici che compongono le word: tutto quello che serve è sempre subito disponibile davanti ai propri occhi.

Il FORTH permette un facile accesso al linguaggio della CPU, possibilità decisamente limitata in altri linguaggi. Si possono indirizzare tutte le celle di memoria e di I/O e vi sono tutte le word pronte per fare tutto quello che normalmente possono fare le varie istruzioni del linguaggio macchina. Se poi alcune operazioni necessitano di una velocità di esecuzione estrema, ecco che il FORTH permette la stesura di word direttamente in linguaggio Assembly, dato che il FORTH dell'RSC-65F11 contiene persino un Assembler. Quest'ultimo è ad un solo passo ed è implementato in circa 1,5 Kbyte nella ROM di sviluppo (si veda più avanti), e pure questa compattezza la dice lunga sulla potenza del linguaggio. Le word appena scritte, che pure contengono istruzioni in linguaggio macchina, vengono viste dal sistema operativo né più né meno come le altre, per cui possono essere testate immediatamente e messe via per dopo. Addirittura la programmazione strutturata è incoraggiata perfino con l'Assembler: esso infatti contiene già le macro-definizioni IF...ELSE e BEGIN...UNTIL, che naturalmente possono essere mischiate alle istruzioni Assembly. Alla fine, comunque, il programmatore può anche dimenticarsi quali sono le word scritte in linguaggio macchina, perché, come ho già detto, per il sistema operativo non esiste alcuna differenza né vi è alcun problema di rilocabilità o connessione con le altre word. Così un programma complesso può essere scritto tutto in alto livello e compilato; poi, se si sente la necessità di accelerare l'esecuzione di alcune parti, le relative word possono essere riscritte in linguaggio macchina senza che il programmatore debba preoccuparsi di apportare modifiche da nessuna parte.

Il FORTH è estremamente trasportabile da macchina a macchina. È del resto una pratica comune per grossi programmi essere sviluppati ad esempio su un PDP-11 e poi lavorare su Z80 o 6502 senza modifiche o quasi. Anche il FORTH del nostro RSC-65F11 segue il modello del FIG (FORTH Interest Group), che è il più comune "dialetto" del FORTH e quello più consono ai dettami dello standard internazionale. Gli utenti del nostro sistema potranno quindi anche scrivere programmi in FORTH, che poi gireranno su altre macchine o, viceversa, altri programmi potranno girare senza difficoltà sul nostro sistema. Di solito, infatti, i programmi non contengono parti in linguaggio macchina, ma sono scritti interamente in FORTH, però in modo che le parti più critiche, con piccolo sforzo, possano eventualmente essere riadattate con nuove word in linguaggio macchina.

Il debugging

Nell'ambiente FORTH la facilità di verifica e di controllo degli errori di moduli sono importanti vantaggi offerti dal sistema operativo. Viene permesso un completo accesso alla macchina, al contrario di ciò che avviene con BASIC, Pascal o simili linguaggi, che invece cercano di proteggere il programmatore contro gli errori. Nonostante ciò i programmi realizzati sono molto affidabili e, cosa non certo di poca importanza, anche facili da modificare in passi successivi. Il FORTH sta all'erta contro eventuali errori sia durante la compilazione che durante l'esecuzione delle word, ma i mezzi più efficaci per il debugging consistono, se così si può dire, nella struttura stessa del linguaggio.

Per quanto detto i programmi FORTH devono essere progettati partendo dal livello di complessità più elevato, scindendolo via via in moduli sempre meno complessi, secondo una struttura ad albero (più o meno ramificata), che descrive tutto il programma. Questo significa procedere top-down. Invece in fase di introduzione si partirà dai moduli a livello più basso, con un metodo chiaramente basso-alto, ossia bottom-up. Si inizierà così dalle word che chiamano solo word già insite nel kernel dell'RSC-65F11. Siccome le avremo scritte in modo che siano corte e facili da testare, potremo provarle ampiamente nei confronti di tutti i possibili casi parametro-in/dato-out. Più una word è semplice e più facile sarà testarla completamente. Come si testa una word? Lo abbiamo già accennato: si predispongono artificialmente dei parametri sullo stack, si esegue la word e poi si torna a rivedere lo stack, verificando che i dati lasciati siano corretti.

Un esempio: si sia creata la word DECIMALE?, la quale deve verificare che un numero sia fra 0 e 9 compresi, ritornando 1, se è vero, 0, se è falso. Dovremo fare il test caricando sullo stack un numero intermedio (esempio: 5), per poi chiamare DECIMALE? e vedere se al posto del numero la word ha lasciato il flag (acceso). Il test dovrà procedere con valori estremi e fuori dall'intervallo, ed in tutti i casi sarà assai importante verificare che la word non alteri altre celle dello stack all'infuori di quelle prefissate. Anche questa verifica è facile: basterà infatti pre-caricare lo

NOME DEL SEGNALE	PIEDINO	DESCRIZIONE
V_{cc}	21	Alimentazione del chip (+5V).
V_{rr}	39	Alimentazione della RAM in tampone.
V_{ss}	40	Massa.
XTLI	2	Terminale del quarzo (se si usa l'oscillatore interno), oppure ingresso di clock (con oscillatore esterno).
XTLO	1	Secondo terminale del quarzo (oscillatore interno).
RESET	20	Segnale di Reset per la CPU. Tale segnale deve essere tenuto a 0 logico per almeno 8 cicli del clock dopo che è applicata la V_{cc} al sistema.
FASE 2	3	Segnale di clock di sistema.
NMI-negato	22	Un impulso da 1 a 0 su questo piedino segnala un interrupt non mascherabile alla logica interna della CPU.
PA0...PA7 PB0...PB7 PC0...PC7: A0...A3, A12 R/W, A13 EMS-negato	30...23 38...31 4...11	Due port di I/O disponibili per collegamenti a dispositivi esterni al chip. Il port C viene usato per presentare all'esterno le linee di indirizzo e di controllo (R/W) verso i dispositivi esterni alla CPU.
PD0...PD7: A4...A11 D0...D7	19...12	Il port D viene usato per presentare all'esterno le linee di indirizzo e di dato, in forma multiplexata. Il cambiamento è indicato dalla transizione del segnale EMS-negato.

Figura 4 - Funzioni dei piedini del chip RSC-65F11.

Costruiamo una FORTH-Card con il chip RSC-65F11

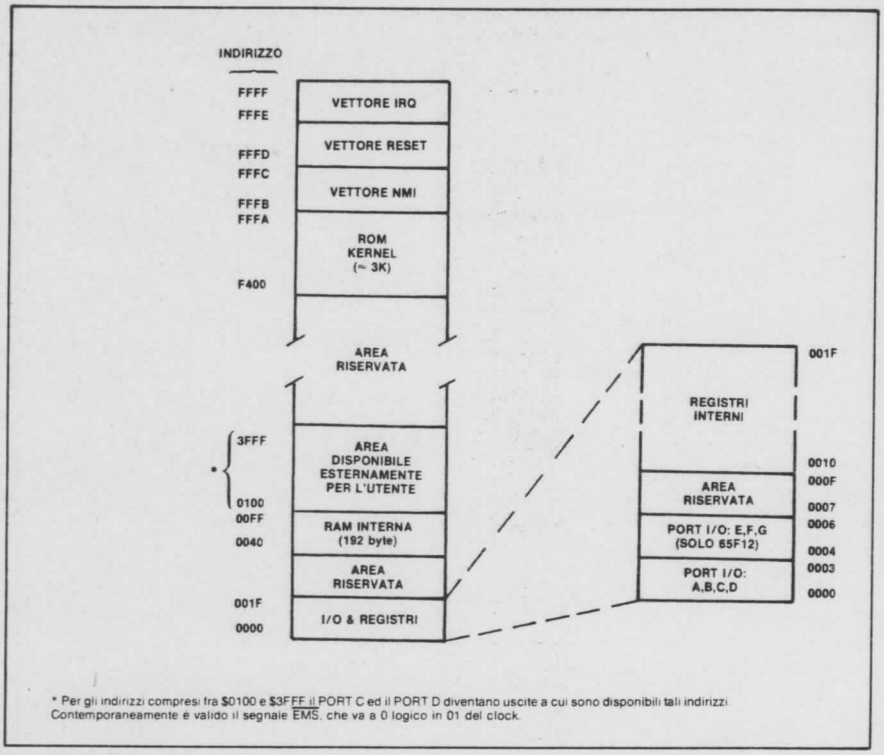


Figura 5 - Mappa di memoria del single-chip RSC-65F11.

stack con una sequenza nota di valori (esempio: 1,2,3,4,5). Il parametro usato dalla word sarà quindi immesso sopra tale sequenza. Dopo l'esecuzione della word si verifica lo stack e se la sequenza è intatta vuol dire che la word non produce effetti di sconfinamento su altre zone dello stack.
Terminate le operazioni di test sulla word si aggiungerà la nuova definizione nel proprio glossario, la quale sarà quindi la documentazione che costituirà il programma. Un buon glossario può essere impostato lungo i passi seguenti:
- definire il nome della word;
- definire chiaramente quanti e quali parametri vanno preparati sullo stack prima di chiamare la word;
- definire chiaramente quanti e quali parametri la word lascia sullo stack;
- descrivere brevemente, ma in modo completo, il funzionamento della word.

Forse questo modo di procedere può sembrare strano, o perlomeno diverso dal solito riga di statement + commento; ma il suo uso, ne son sicuro, farà ricredere parecchi scettici.

Anche perché ho l'impressione che finora l'argomento FORTH sia sempre stato trattato o come un curioso ghiribizzo della fantasia o come un linguaggio, interessante, ma destinato a persone diverse dai comuni mortali.
Ebbene, questi articoli cercheranno di far toccare con mano il FORTH con l'ausilio di un sistema vero, autonomo e potente, anche se miniaturizzato. Soprattutto non ci saranno più dubbi sulle possibilità applicative del FORTH. Non certo quello del tipo: "quando ho il programma che gira sull'Apple o sul Commodore, non posso mica destinare il mio personal definitivamente per l'applicazione specifica!".
Con la nostra FORTH-Card, che è anche e soprattutto un microcontroller autonomo, tutti questi problemi spariscono, perché si avrà un sistema indipendente ed economico che potrà essere dedicato ad una specifica applicazione, come e quando lo si vorrà.
Concludendo la panoramica sul FORTH, e sui suoi aspetti più caratteristici, posso senz'altro anticipare che un piccolo "tutorial" sull'uso hardware e software della FORTH-Card seguirà nei prossimi articoli; ora infatti voglio destinare lo spazio rimanente alla descrizione funzionale del chip RSC-65F11 ed alla impostazione dello schema-base della FORTH-Card stessa.

Il single-chip RSC-65F11

La figura 2 mostra la struttura interna dell'RSC-65F11.
La CPU è un modello potenziato dell'oramai famosa 6502, con un set di istruzioni arricchito di quattro tipi, utilissimi in programmi di controllo. Le istruzioni aggiunte sono:
- BBR cioè Branch on Bit Reset;
- BBS cioè Branch on Bit Set;
- SMB cioè Set Memory Bit;
- RMB cioè Reset Memory Bit.

Come si vede, sono istruzioni ottime nella manipolazione dei bit di un byte ed accelerano di circa un 10% molte routine del nucleo FORTH dello stesso RSC-65F11.
Per il resto la CPU è la classica 6502 coi suoi registri A, X, Y, lo Stato e lo Stack Pointer a 8 bit e naturalmente il Program Counter a 16 bit.

Oltre alla CPU vi sono diversi blocchi di supporto, che rendono completamente autonomo il microcomputer.
Innanzitutto la ROM, che contiene, da $F400 a $FFFF, i vettori di RESET, NMI ed IRQ ed ovviamente anche il nucleo (kernel) del FORTH. La figura 5 mostra in dettaglio la mappa di memoria nel modo multiplexato. Si noti che non tutti i 64 Kbyte indirizzabili sono disponibili all'esterno, in quanto il single-chip già contiene una decodifica precisa, che rende disponibile solo un quarto della mappa per l'utente. Dati ed indirizzi sono emessi in un modo un po' strano, usando gli stessi port di I/O (port D e port C) previsti nella versione più generale del chip (la 6511Q). Nel 65F11, grazie a una circuiteria interna aggiunta appositamente, il ciclomacchina viene gestito come segue (la discussione è facilitata dal diagramma temporale di figura 6):
- un ciclo, della durata di un periodo del clock di sistema, inizia con una semionda a zero logico, detta anche fase 1;
- dopo il tempo Tpcas gli indirizzi sono stabili e disponibili per i dispositivi esterni;
- dopo Tesu, invece, un apposito segnale, detto EMS-negato (che sta per "external memory select"), da 1 va a 0 e segnala che occorre memorizzare una parte del bus-indirizzi (da A4 ad A11), perché poi tali indirizzi spariranno per essere sostituiti, sulle stesse linee, dai dati. Se sfruttiamo tale segnale EMS come abilitazione per un latch del tipo SN74LS373, potremo agevolmente memorizzare i suddetti otto indirizzi, rendendoli stabili per l'uso mediante dispositivi esterni;
- quando EMS è andato a 0 gli indirizzi multiplexati restano sul bus ancora per Tpcvd, quindi vengono sostituiti con i dati. Se questi ultimi, per una scrittura, arrivano dall'RSC-65F11, essi saranno disponibili Tpbdd nanosecondi dopo l'inizio della semionda positiva del clock (fase 2);
- nel caso di una lettura, invece, i dati devono essere emessi dal dispositivo esterno (esempio: memoria) almeno Tpbsu prima della fine della semionda positiva.

Da questa analisi consegue che si possono usare tutti i più comuni tipi di memorie RAM, ROM o EPROM, avendo però l'ac-

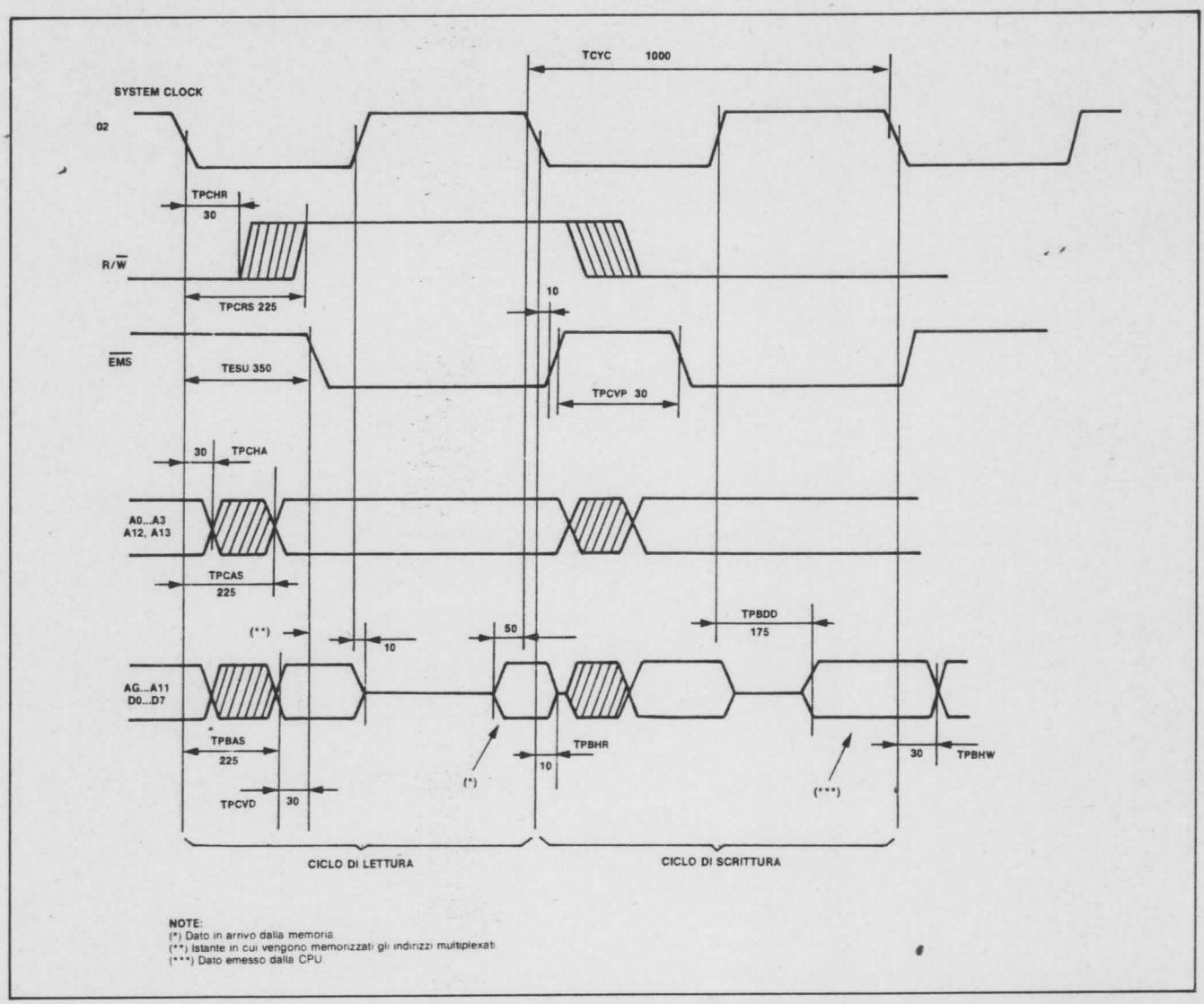


Figura 6 - Diagramma temporale di due cicli-macchina dell'RSC-65F11: un ciclo di lettura seguito da un ciclo di scrittura. I tempi evidenziati indicano i valori tipici in nanosecondi. Si presuppone una frequenza di clock di 1 MHz.

cortezza di generare i segnali di accesso, come OE-negato e WE-negato (cosa assai importante soprattutto per le RAM), solo nella fase 2.
Per il resto nessun problema, anche perché tutti gli altri dispositivi sono interni al single-chip e non vi è da preoccuparsi delle questioni hardware.
Va notato (figura 5) che il segnale EMS è valido solo per i primi 16 Kbyte indirizzabili, eccettuata la pagina zero, ove nell'RSC-65F11 sono condensati gli I/O, i registri interni e - nei famosi 192 byte di RAM integrati - la "vera" pagina zero e lo stack della CPU, che quindi non è in pagina 1 come in una 6502 normale.
Da $00FF in su l'utente può indirizzare i suoi chip di espansione con una mappa, che tipicamente sarà:

- da $0100 a $01FF: floppy disk controller;
- da $0200 a $1FFF: circa 8 Kbyte di RAM;
- da $2000 a $3FFF: EPROM o ROM con il programma.

Tale mappa è solo consigliata, ma può senz'altro essere ridotta, soprattutto per applicazioni minime, e la figura 7, che finalmente mostra lo schema più semplice realizzabile, indica infatti che già con soli 2 Kbyte di RAM e la ROM di sviluppo si può iniziare a lavorare.
Ancora, per gli aficionados dell'hardware, qualche considerazione su quest'ultima figura. Come si noterà, abbiamo messo il latch 74LS373 per catturare i bit d'indirizzo intermedi; questi, assieme agli altri non multiplexati, sono inviati ad una decodifica ultrasemplice, abilitata solo in fase 2, ed alle due memorie.
La prima è una normalissima RAM da 2 Kbyte x 8 byte, mentre la seconda è la ROM 65FR1, che contiene una ulteriore aggiunta al nucleo originale del FORTH e permette di tramutare questo gruppetto di soli cinque chip in un vero e proprio sistema di sviluppo in FORTH. Per non appesantire ora la discussione, della 65FR1 parleremo dopo.
Rientriamo per un momento nel single-chip e riprendiamo la figura 2: non dobbiamo infatti dimenticare che nell'RSC-65F11 vi sono parecchi dispositivi accessori, oltre alla CPU.
Unitamente a quelli di I/O, con funzioni molto simili a quelle di una VIA 6522, vi sono pure due timer/counter a 16 bit (anch'essi funzionano e si controllano come quelli di una VIA 6522), ed infine sul

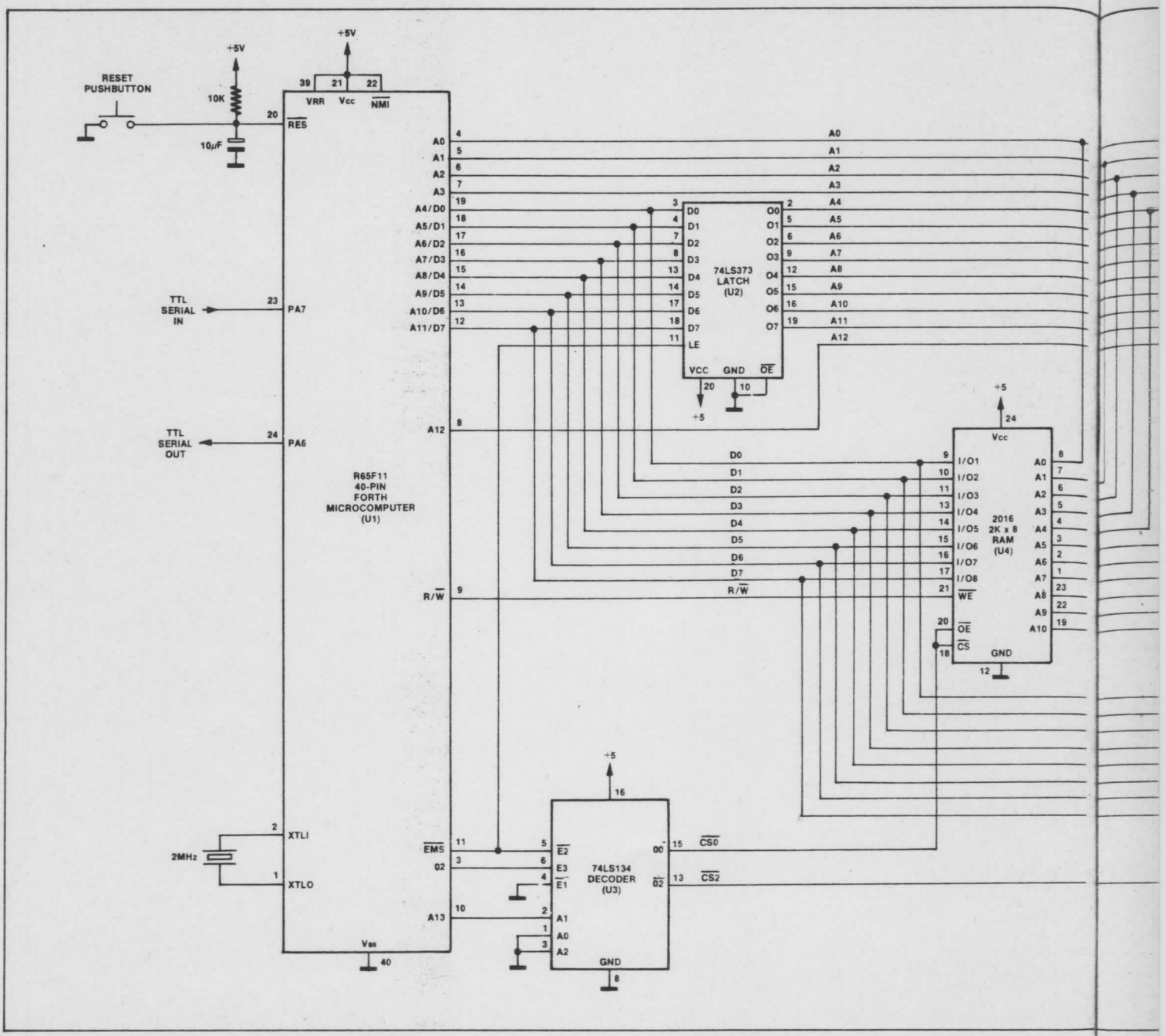


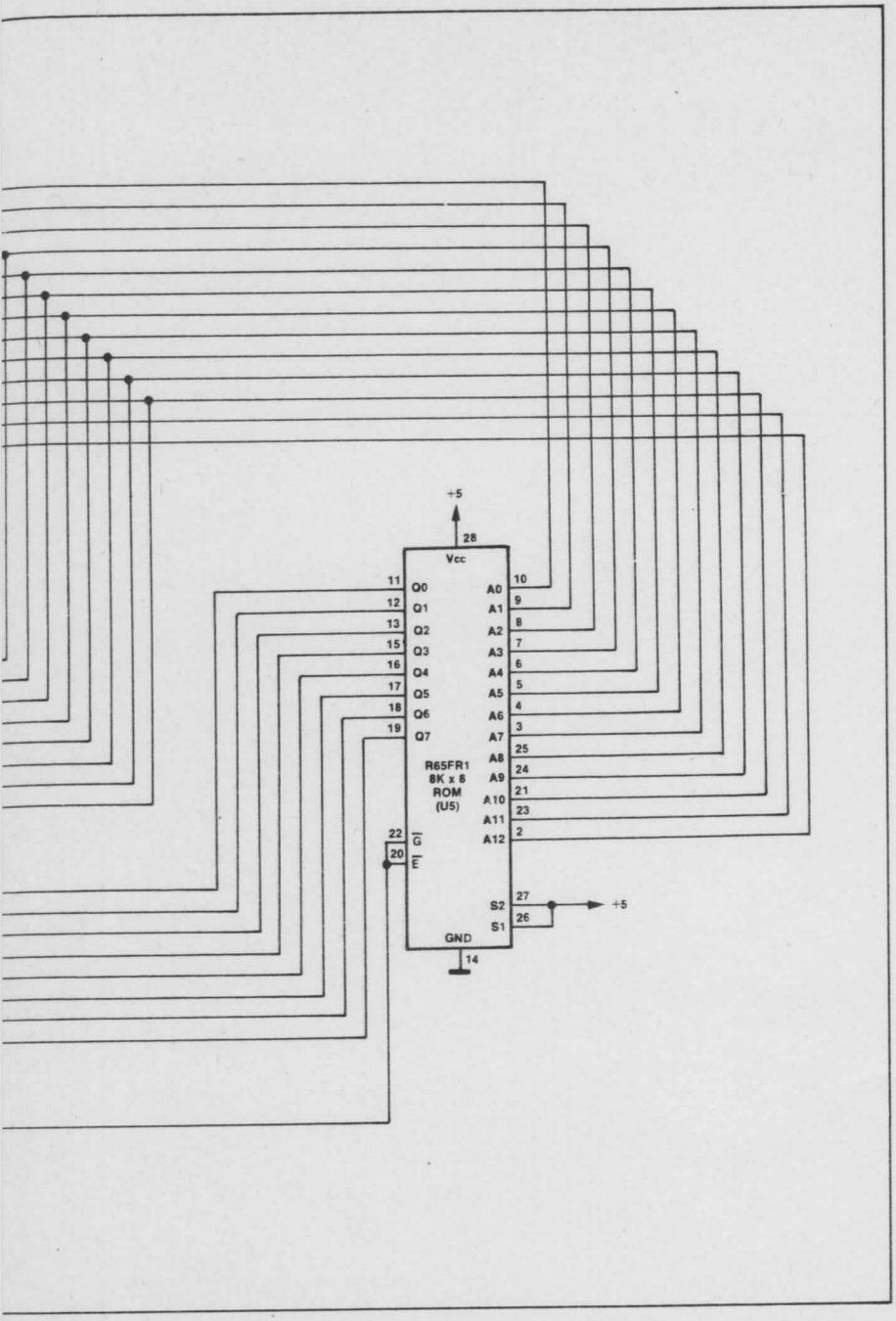
Figura 7 - Con solo cinque componenti è possibile realizzare un circuito per la valutazione del chip RSC-65F11. Vedremo nel prossimo al contempo assai più versatile, che comprende, volendo, anche una completa interfaccia per dischi da 5"1/4.

chip è integrata un'unità di trasmissione-/ricezione seriale, con baud rate programmabile via software. Se al posto dell'RSC-65F11 si lavora con il chip RSC-65F12, si hanno altri tre port di I/O in più. Non abbiamo preso in considerazione, per il progetto, la versione maggiorata del chip, perché costa di più e richiede zoccoli particolari, trattandosi di un package da 64 piedini in configurazione quip (quad-in-line package). Quello dell'RSC-65F11 è un normale contenitore da 40 piedini.

Descrizione funzionale del sistema

Dopo aver accontentato i patiti dell'hardware, siamo giunti al momento in cui entra in scena, finalmente, sua maestà il Sistema: ovvero la circuiteria basata non solo sull'RSC-65F11, ma anche sui chip esterni ad esso collegati. Come funziona questo mini-sistema?

Prendendo come spunto la figura 7, si nota innanzitutto che le due linee seriali devono obbligatoriamente essere collegate a un terminale, perché la FORTH-



articolo, invece, che da tale schema si è elaborato un circuito lievemente più complesso, ma

Card, almeno nella fase di sviluppo di un programma, colloquia interattivamente con l'utente, grazie alla linea seriale full-duplex. I piedini 23 e 24, naturalmente, accettano segnali a livello TTL, per cui, se il terminale usato trasmette in RS232-C, occorre usare un ricevitore del tipo 1489 sulla linea RX.

La decodifica è spartana ed è abilitata solo nella fase 2 del clock di sistema, e solo quando l'EMS è valido (a 0 logico). Sono usati due segnali di selezione: CS0-negato chiama la RAM, ed è ampio 8 Kbyte (anche se la RAM è di soli 2 Kbyte); CS2-negato chiama invece la ROM di sviluppo (che è veramente di 8 Kbyte). Con una simile circuiteria si può convenientemente provare il chip RSC-65F11, però consiglierei di attendere la descrizione della FORTH-Card vera e propria, nel prossimo articolo: il circuito sarà un pochino più complesso, ma la versatilità ne guadagnerà moltissimo.

Che cos'è la ROM 65FR1, detta "di sviluppo"? Ebbene, si tratta semplicemente di un programma, scritto con le word del kernel dell'RSC-65F11, grazie al quale è possibile gestire interattivamente lo sviluppo dei programmi. La 65FR1 aggiunge ben 153 nuove word alle originali 133 del nucleo ROM nell'RSC-65F11. Solo se è presente la ROM di sviluppo si può dialogare ad alto livello col sistema, mentre senza tale ROM è comunque possibile un dialogo ridotto in ambiente monitor (ebbene sì, anche se molto spartano, c'è anche un micro monitor di sistema!).

Accensione del sistema

Una volta collegato il terminale alla FORTH-Card si può dare tensione al sistema. Il circuito di reset (che in figura 7 è un semplice condensatore + resistore) si attiva e, quando la linea di reset torna alta, il vettore di reset nella ROM del kernel punta ad una word scritta interamente in codice macchina, il cui nome è COLD (in inglese: freddo). La word COLD inizializza il microcomputer in tutte le sue parti: registri di controllo, port di I/O, canale seriale. Quest'ultimo è programmato per comunicare a 1200 baud, in modo asincrono, con sette bit di dato, nessun bit di parità e due bit di stop. Poi altre variabili di sistema sono inizializzate con l'ausilio di tabelle già pronte in ROM.

A questo punto viene eseguito un test per vedere se è la prima accensione (cold start) o se il reset avviene a macchina già accesa (warm start). Il test è eseguito verificando se la variabile CLD/WRM all'indirizzo $030E:$030F contiene il valore $A55A. Se così non è, viene eseguita una inizializzazione completa (cold start). Durante tale inizializzazione i vettori di IRQ ed NMI vengono fatti puntare alla stessa routine di reset per evitare che eventuali chiamate di interrupt blocchino il sistema; poi vengono inizializzate tutte le altre variabili di sistema, fra cui la CLD/WRM con $A55A, e solo a questo punto la FORTH-Card è pronta a ricevere comandi dall'utente.

Se, invece, la variabile CLD/WRM contiene $A55A, solo alcune variabili vengono rimesse ai loro valori di default e questo, tra l'altro, evita che l'utente perda dal dizionario tutte le definizioni appena compilate.

Un altro caso, interessante da analizzare per gli sviluppi futuri della FORTH-Card, riguarda un programma cosiddetto di auto-start. Infatti, sia che venga eseguito un reset cold che un reset warm, la mappa di memoria, da $0400 in su e ad inter-

valli di 1 Kbyte (cioè: $0400, $0800 e così via), viene esaminata per trovare una sequenza del tipo $A55A. Se i primi due byte del blocco di 1024 contengono questa sequenza, si assume che tutto ciò che segue sia il contenuto di una ROM con un programma FORTH, che deve essere subito fatto partire. Perché ciò sia possibile i due byte successivi alla sequenza-chiave devono contenere un indirizzo che punta alla prima word del programma auto-start, ovverosia alla word di livello più alto, che sarà in pratica il nome del programma stesso.

La stessa ROM 65FR1 è un esempio di ROM, che contiene un programma auto-start: è il programma che permette all'utente di comunicare interattivamente col sistema operativo e di sviluppare i suoi programmi.

Se dal sistema si toglie tale ROM di sviluppo, lasciando la RAM, all'accensione la routine di reset non trova alcuna chiave $A55A, si accorge cioè che non esiste nessuna ROM con un programma auto-start. In tal caso viene emesso un messaggio sul canale seriale: NO ROM, per avvisare l'utente che non esiste alcun programma auto-start in memoria.

Il Micro-Monitor

L'utente può tuttavia colloquiare ancora col sistema operativo, anche senza la ROM di sviluppo: battendo infatti a terminale un CONTROL-R, si potrà entrare nel Micro-Monitor del FORTH, che segnala la sua disponibilità a ricevere comandi emettendo il prompt ">". In ambiente Micro-Monitor (d'ora in poi: MM) sono ammessi solo due comandi, seguiti da un parametro opportuno:

- il comando "N", seguito da un numero, fa sì che quest'ultimo sia interpretato come un valore esadecimale a 16 bit da porre in cima allo stack;
- il comando "W", seguito da un indirizzo esadecimale, fa sì che venga eseguita immediatamente la word il cui puntatore (PFA) è rappresentato da quell'indirizzo.

Naturalmente, occorre conoscere tutti i PFA delle word del kernel, cosa che il manuale della FORTH-Card documenta ampiamente, assieme a tutti i PFA delle word della ROM di sviluppo 65FR1.

In ambiente MM un solo comando può essere immesso su ogni riga. Il comando va fatto seguire da un RETURN da terminale per avviarne l'esecuzione.

Qualche esempio:

N1 < cr >
N2 < cr > (ho messo i valori 1 e 2 nello stack)
WF778 <cr > ($F778 è il PFA della word

GLOSSARIO

Baud rate: è la velocità di trasmissione dei dati su un canale seriale, che si misura in bit/s.

Stack (catasta): è un'area della RAM di sistema, ove vengono riposti i dati, che possono rappresentare indirizzi o valori numerici. Uno stack è definito solo quando vi è un "puntatore" (di stack) che contiene l'indirizzo dell'ultima casella di memoria occupata: alla memorizzazione successiva, infatti, il puntatore sarà aggiornato per puntare alla prossima cella contigua all'ultima appena occupata. Se per esempio, come è evidenziato nella figura 8, si attribuisce allo stack la zona da $0200 a $02FF, si inizierà con lo stack vuoto ed il suo puntatore - a 16 bit - che contiene $01FF: (lo stack allora cresce verso l'alto della memoria). Se ora salviamo un dato di 8 bit nello stack, lo metteremo all'indirizzo $0200, cioè nella prima cella libera, e lo stack pointer sarà aggiornato a $0200. Se salviamo un altro valore, ma di 16 bit, lo metteremo in $0201 e in $0202, e questo sarà anche l'ultimo valore aggiornato del puntatore. Infine, se ripeschiamo dallo stack un valore di 8 bit, prenderemo quello in $0202 ed il puntatore diventerà $0201. In definitiva, lo stack è una memoria LIFO (Last In First Out) dove il dato che entra per ultimo viene anche ripescato per primo. L'utilità dello stack è evidente nello scambio dei parametri fra due parti del programma: non c'è infatti più bisogno di definire zone riservate per tali parametri, perché basta che la prima subroutine lasci nello stack l'esatto numero di dati che servono alla seconda subroutine e questa li userà, mentre nel frattempo lo stack funziona come un soffietto. Su questa idea, in pratica, è basato tutto il funzionamento del FORTH.

Ricursività: una subroutine è ricursiva quando, ad un certo punto della sua esecuzione, chiama se stessa. Naturalmente, se si vorranno evitare loop senza fine, la chiamata dovrà essere effettuata con parametri differenti nello stack.

Top-down: è un metodo di analisi e di sviluppo di un problema che consiste nell'approfondimento per fasi successive del problema stesso, fino a definire, negli ultimi passi, i dettagli più particolareggiati. Si veda anche strutturazione.

Bottom-up: è il metodo esattamente contrario al top-down. Si parte definendo i particolari e via via si sale di complessità fino a definire tutto il problema.

Strutturazione: è la metodologia d'impostazione di un programma, per cui si affronta un problema con metodo top-down (vedi). In termini software vi saranno livelli sempre più bassi di subroutine, senza alcun salto (GOTO) fra i vari pezzi del programma. È tipico dei programmi scritti in Pascal, Modula-2, ADA e, naturalmente, FORTH.

Glossario: non ha nulla a che vedere con "questo" glossario! È invece il metodo di documentazione delle subroutine (word) del FORTH, per cui il programmatore descrive con completezza una word, definendo quali parametri usa e come funziona. Alla fine del lavoro fatto per tutte le word si ottiene una sorta di mini-enciclopedia che si chiama, giustappunto, glossario.

Linguaggio macchina (anche: linguaggio Assembly): è il linguaggio che definisce le istruzioni della CPU del sistema, per cui vi è corrispondenza biunivoca fra l'istruzione in linguaggio macchina ed il codice binario che la rappresenta e che sarà eseguito dalla CPU. Esempi di tali istruzioni possono essere: LDA dato, STA indirizzo, JMP

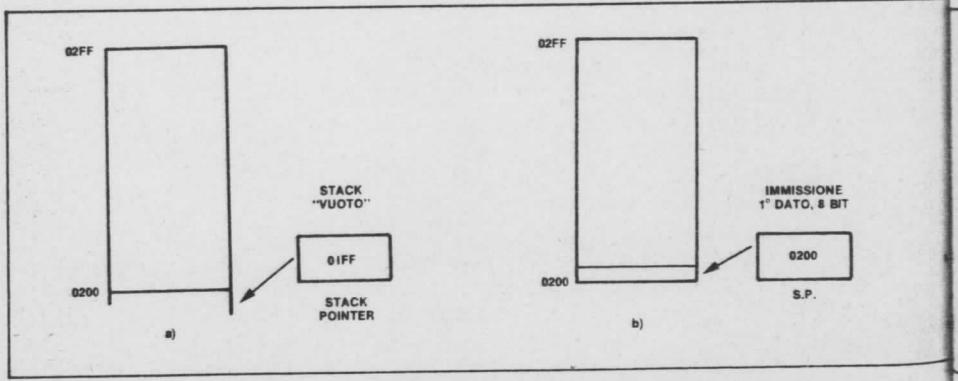
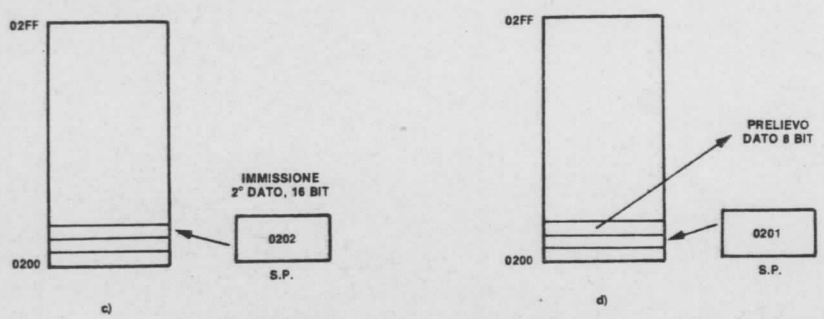


Figura 8 - Tale diagramma, che si riferisce alla discussione sullo Stack presente nel GLOSSARIO, illustra le variazioni sia dello Stack che del relativo Stack Pointer quando

indirizzo, ecc.
Linguaggio Assembly: vedi anche linguaggio macchina.
Linguaggio ad alto livello: è il linguaggio le cui istruzioni non sono più legate alla struttura della CPU, ma seguono una sintassi predefinita. In un linguaggio ad alto livello le singole istruzioni solitamente sintetizzano molte istruzioni del linguaggio macchina, per cui è più facile scrivere un programma, ad esempio, in BASIC, piuttosto che in Assembly.
Compilatore: è un programma nel calcolatore che trasforma un altro programma, scritto in linguaggio ad alto livello, in un equivalente programma in linguaggio macchina, dato che solo le istruzioni di quest'ultimo corrispondono a quelle della CPU e quindi potranno essere eseguite.
Interprete: è un programma che trasforma, in passi successivi, le varie istruzioni di un programma scritto in linguaggio ad alto livello in particolari codici "intermedi", che, durante la successiva esecuzione, saranno uno ad uno tradotti sul momento in istruzioni macchina eseguibili dalla CPU. Quindi la esecuzione di un programma interpretato implica, ogni volta, una continua e contemporanea traduzione dal codice intermedio al codice macchina, mentre un programma compilato è tradotto subito, e per intero, in codici macchina definitivi. L'esecuzione di un programma compilato è quindi assai più veloce di quella di un programma interpretato.
Interattivo: si chiama così un programma che ha bisogno di un colloquio con l'utente (tramite terminale video, solitamente) per proseguire nelle sue funzioni o, come nel caso di un linguaggio, per accettare ed eseguire comandi.
Sistema operativo: è il primo e più importante dei programmi che risiedono nel calcolatore, dato che è quello che viene attivato subito dopo l'accensione del sistema, per cui ha il compito di predisporre tutte le varie parti del computer (inizializzazione) e di permettere il successivo uso del computer da parte dell'utente.
T.P.I.: "Tracks Per Inch", ossia "tracce per pollice". È il numero di tracce utili per la registrazione su un disco, ovviamente nella unità di misura inglese. Attualmente vi sono drive che possono registrare dischi a 96 T.P.I. o dischi a 48 T.P.I.
Memoria virtuale: è un metodo di gestione delle risorse di memoria RAM del computer in simbiosi con una unità di memoria di massa, tipicamente dei dischi. Sia pure penalizzando la rapidità, si realizza un notevole risparmio della memoria "vera" mappata sul bus. La tecnica base è relativamente semplice: il programmatore, per i suoi dati e/o programmi, usa una ben definita zona di RAM che, quando è piena, viene automaticamente ricopiata su disco. Nel caso di letture la RAM viene svuotata dall'utente e, quando è vuota, è di nuovo riempita caricando i dati da disco, sempre in modo automatico.
Bus multiplexato: è tipico di parecchie CPU che usano un certo numero di piedini del contenitore ("package") per diverse funzioni. Nell'RSC-65F11, ad esempio, i piedini dal 12 al 19 servono sia per gli indirizzi (A4..A11) che per i dati (D0..D7). Se il bus è multiplexato, esiste sempre un segnale particolare, su un altro piedino, che indica quando quei piedini cambiano funzione. In tal modo l'elettronica a valle della CPU può sincronizzarsi rispetto ai cambiamenti di questo bus. Nell'RSC-65F11 tale segnale è l'EMS-negato.



nell'area di Stack vengono immessi o prelevati dei dati.

"+")

WFEE4 < cr > ($FEE4 è il PFA della word "." che stampa il numero in cima allo stack del FORTH)
3 (è il risultato atteso)

Anche se estremamente spartano, il MM permette di lavorare col FORTH, per test "on-field", avendo a disposizione tutta la potenza del kernel dell'RSC-65F11; lascio quindi al lettore il compito di immaginarne i mille possibili usi.

Accensione della FORTH-Card con la ROM 65FR1 (e anticipazioni sul seguito)

Se sulla FORTH-Card è installata la ROM 65FR1, all'accensione il terminale evidenzierà:

RSC-FORTH V1.5

È questo il "ciao" del sistema operativo FORTH. Si potrà allora iniziare a lavorare con la scheda, e, di solito, la prima cosa che si chiede è una lista di tutte le word disponibili:

VLIST < cr >

Sul video comincerà a scorrere l'elenco di tutte le 286 word sia del kernel che della ROM 65FR1... ma fermiamoci qui. Lo spazio tiranneggia, così il discorso tutorial sull'uso della FORTH-Card, una volta che sia stata montata, è riservato ai prossimi articoli.
Di tali articoli ne sono previsti almeno due relativi alle modalità d'uso della FORTH-Card e naturalmente del FORTH dell'RSC-65F11. Si partirà dall'esame delle operazioni più elementari, per poi passare ad operazioni più complesse e legate soprattutto alla preparazione di un'interfaccia per dischi flessibili e all'elaborazione di un programma applicativo autostart. Sarà analizzato anche l'assemblatore compreso nel FORTH dell'RSC-65F11 e, naturalmente, studieremo abbastanza a fondo l'hardware della FORTH-Card, nella versione "di consumo".

Io sono già al lavoro per il completamento di questa versione definitiva della FORTH-Card, il cui costo indicativo, per un pezzo singolo, partirà da Lire 300.000 in su, a seconda della estensione RAM, ROM e Floppy Controller. Sto anche lavorando per un protocollo di comunicazione che sfrutti un Apple (con interfaccia seriale), evitando così ai suoi possessori di impegnarsi con un terminale video a sé stante. In maniera approssimativa, dunque, questi saranno gli argomenti che tratterò su **Bit** nei prossimi articoli. ■

Costruiamo una FORTH Card con il chip RSC-65F11

Parte seconda

di **Paolo Bozzola**

Computerjob Elettronica - Brescia

Le novità

I tempi editoriali di una grossa rivista come **Bit** sono piuttosto lunghi e, se c'è di mezzo Agosto, sono ancora più lunghi! Ma in questo caso il periodo estivo è stato utile, perché, dopo le elucubrazioni del primo articolo sulla FORTH Card, abbiamo apportato delle sostanziali modifiche rispetto all'idea di base: per dire i fatti in due parole: il primitivo e spartanissimo prototipo V0.0 (consigliato dalla Rockwell, vedi figura 7 dello scorso articolo) è stato subito abbandonato ed è stato realizzato un secondo prototipo su una basetta preforata in formato doppio-euro. Potete vedere il risultato nella fotografia 1, mentre la fotografia 2 evidenzia l'intrico di connessioni sul retro della piastra (i cavi più grossi sono rinforzi delle linee di massa): non male, vero? Sulla scheda che ammirate nella figura 1 c'è proprio tutto: il single-chip 65F11; tre zoccoli bytewide, di cui due occupati da 8 Kbyte di RAM ed 8 Kbyte di ROM; una completa logica di interfaccia per il bus Minimicro (vedremo dopo che cosa è e quali vantaggi apporta); un collegamento per una stampante tipo Centronics-compatibile ed una interfaccia per floppy disk drive da 5"1/4. Questo prototipo è stato ed è tuttora utilissimo per sperimentare tutta la potenza del linguaggio FORTH implementato col chip 65F11 e, naturalmente, per ottimizzare lo stesso hardware. Fatto ciò siamo passati alla stesura di un terzo progetto, vale a dire la "FORTH Card V2.0", ed è questo progetto che adesso inizieremo a descrivere nei dettagli. Rispetto alla versione V1.0 delle fotografie, la FORTH Card è stata sdoppiata in due schede singolo-euro, cosa che è stata ritenuta più comoda per poterla sfruttare in applicazioni industriali su rack a tre unità; in più sono stati aggiunti due chip di I/O parallelo del tipo VIA 6522. Mentre stiamo scrivendo, questo terzo prototipo è in fase di montaggio. Gli schemi elettrici, naturalmente, esistono già: la figura 1 mostra il circuito della FORTH Card e la figura 2 quello della FORTH-Disk-Card.

Caratteristishe della FORTH Card

Prima di analizzarne a fondo lo schema elettrico, vediamo le principali caratteristiche della FORTH Card nella versione V2.0 qui descritta:

- Unità centrale: single chip RSC 65F11, 6502-based microcomputer con 3 Kbyte di ROM (FORTH Kernel), 16 linee di I/O, 2 Timer/Counter, ingresso ed uscita per comunicazione seriale fino a 62 Kbit/s;
- Memoria: fino a 8 Kbyte di RAM (almeno 6 Kbyte sono necessari per un corretto funzionamento dei dischi); fino a 8 Kbyte di ROM. Sono disponibili tre zoccoli bytewide da 28 piedini, che accettano tutti i più comuni tipi di memorie;
- Clock: 2 MHz esterno, interno 1 MHz, ciclo di istruzione di 1 μs;
- Bus: compatibile Minimicro, con linea di External Address per esclusione automatica della decodifica interna. Indirizzamento possibile a banchi, fino a 4 Mbyte di memoria indirizzabile (48 Kbyte per compatibilità con espansioni Minimicro). Le istruzioni, automaticamente gestite durante l'indirizzamento a banchi, sono BANKC@ (8-bit store), BANKC! (8-bit load), BANKEXECUTE (RUN program) e BANKEEC! (programmazione di EEPROM in un banco);
- Interfaccia seriale: RS-232C, con velocità di 1.200 baud dopo il Reset;
- Collegamento alla stampante: tramite il Port B dello stesso RSC 65F11;
- Dimensioni: Eurocard Standard, 100 x 160 x 25 mm;
- Alimentazione: + 5V e 500 mA; + 12/– 12V @ 50 mA.

Naturalmente, sono sottintese le peculiarità dovute al FORTH, che scopriremo insieme, via via che prenderemo confidenza col sistema.

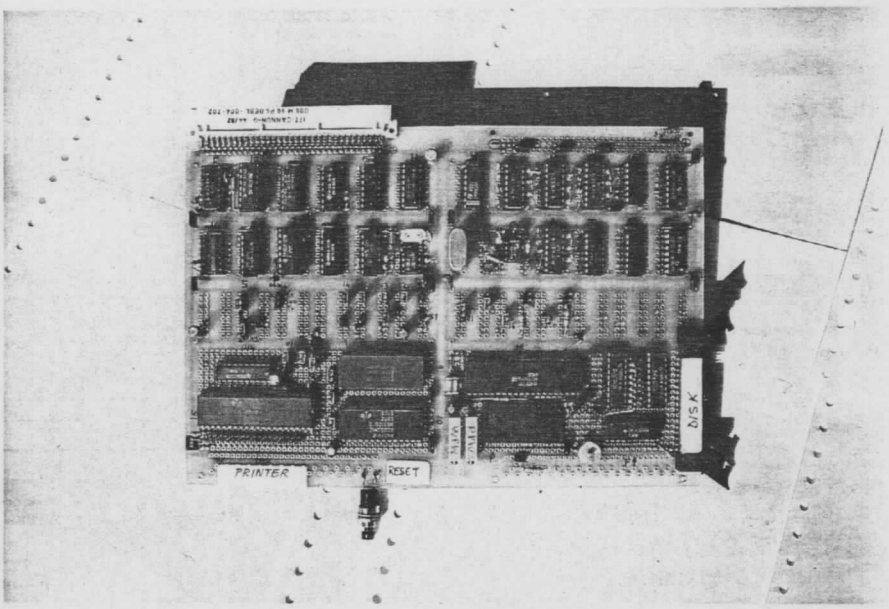


Foto 1 - Il prototipo della FORTH Card, nella sua versione 1.0. È in corso di messa a punto la versione definitiva, su due distinte schede Eurocard.

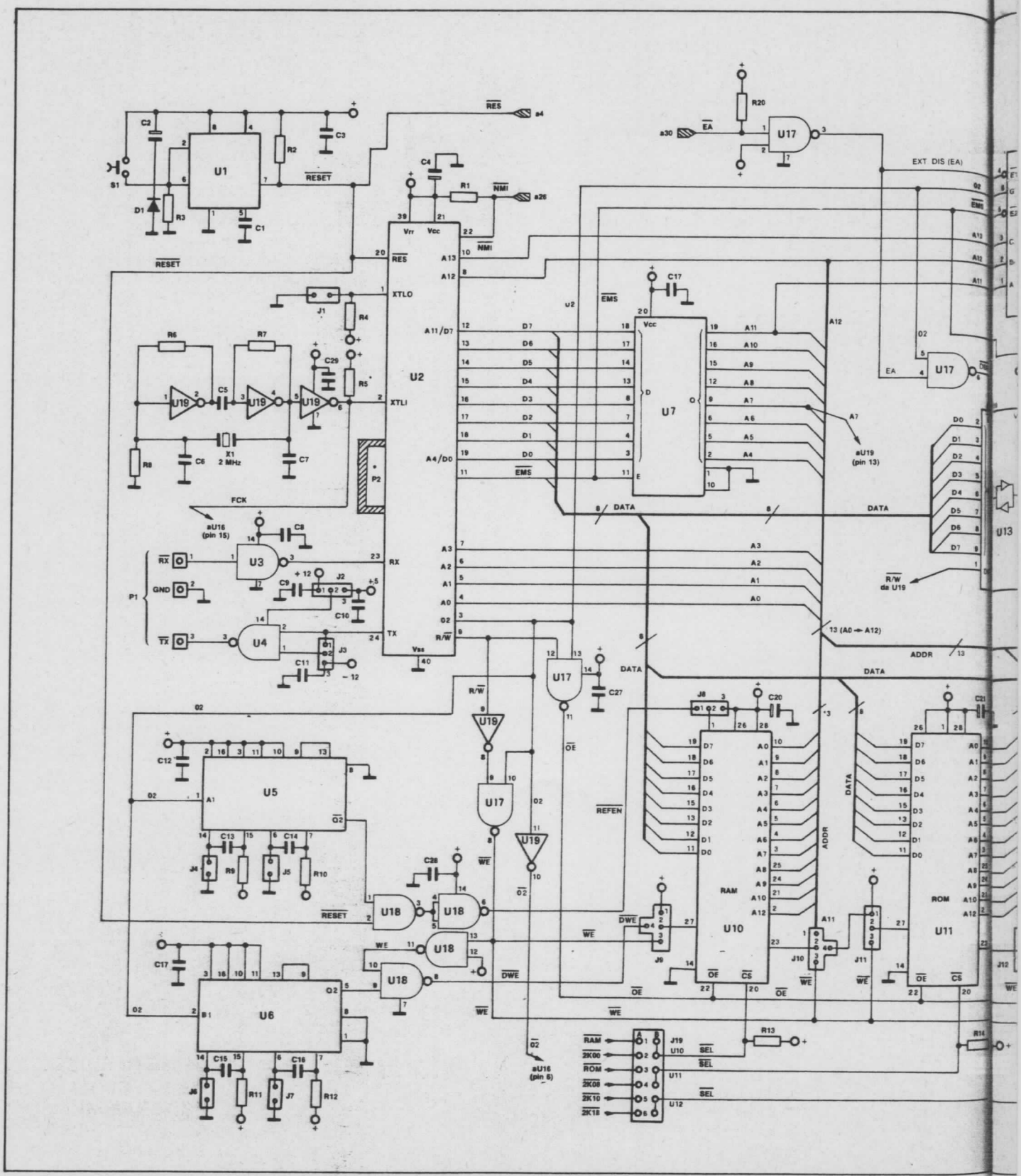
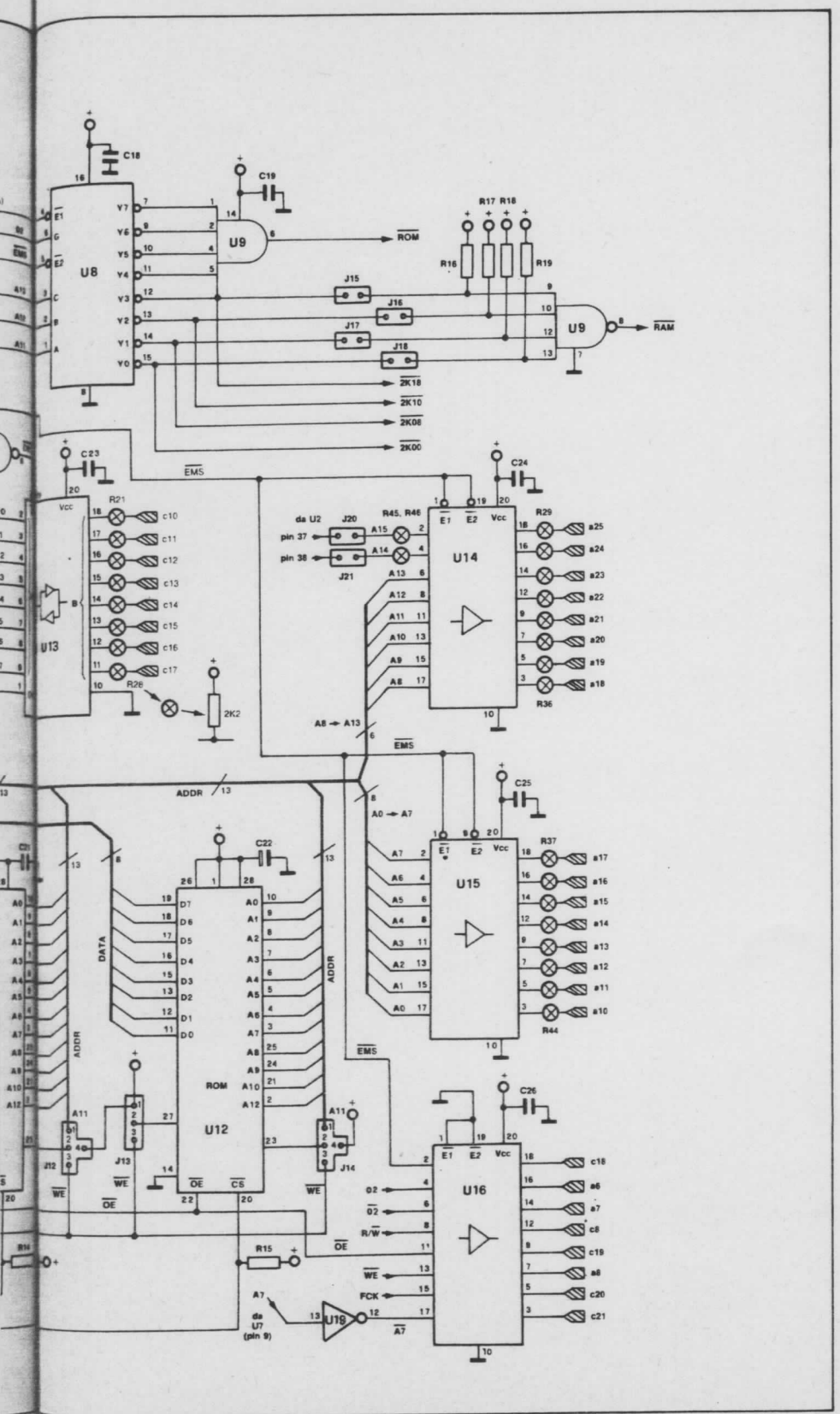


Figura 1 - Schema elettrico della FORTH Card.

Costruiamo una FORTH Card con il chip RSC-65F11



FORTH Card: descrizione dello schema elettrico

La FORTH Card è in pratica la "scheda CPU" del nostro sistema di sviluppo/dimostrativo FORTH. Con riferimento alla figura 1, che ne descrive lo schema elettrico, vediamo che il cuore della scheda è U2, cioè il single-chip 65F11; U1 è il circuito di Reset/Power-On e U19 genera il clock, la cui frequenza è di 2 MHz, in quanto una divisione per due è effettuata internamente al 65F11. Generando 2 MHz, noi possiamo sfruttare tale clock anche per l'interfaccia-dischi (FORTH-Disk-Card). U3 ed U4 gestiscono il collegamento seriale in standard RS-232C, full duplex asincrono: per trasmissioni a breve distanza, a livello TTL normale, U4 può essere sostituito da un semplice 74LS00, previa impostazione dei ponticelli J2 e J3. U7 è il latch che de-multiplexa (separa) i dati dalla parte bassa del bus degli indirizzi: la sua azione è controllata dal segnale EMS-negato, come del resto è stato descritto nel primo articolo di questa serie. La decodifica di sistema è assai semplice, basandosi infatti sul solo U8, un decoder 3→8 del tipo 74LS138. Si noti che sono disponibili, verso l'esterno del chip RSC 65F11, solo 14 linee di indirizzo: e infatti solo 16 Kbyte sono disponibili per l'utente, come ben si nota dalla mappa

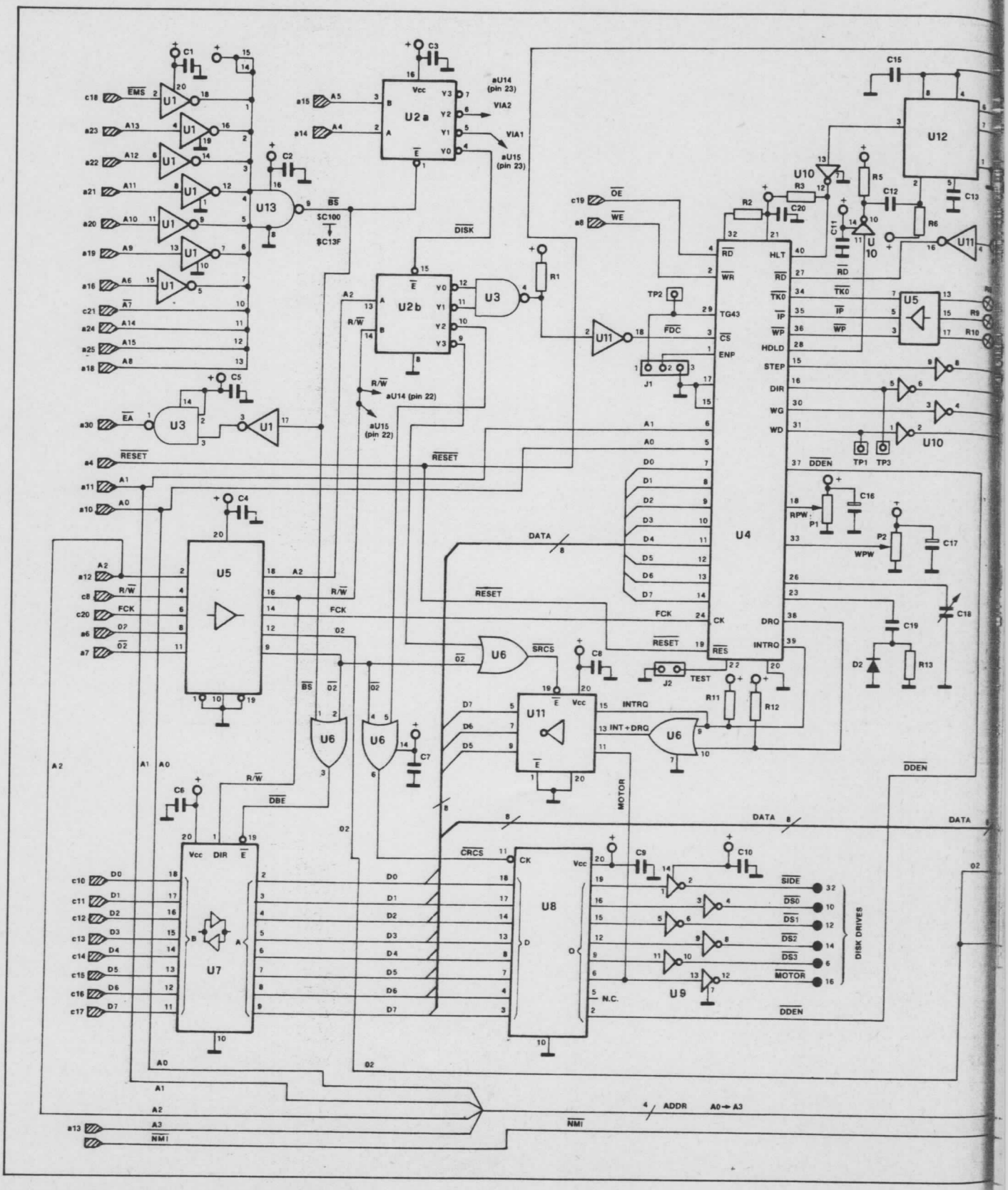
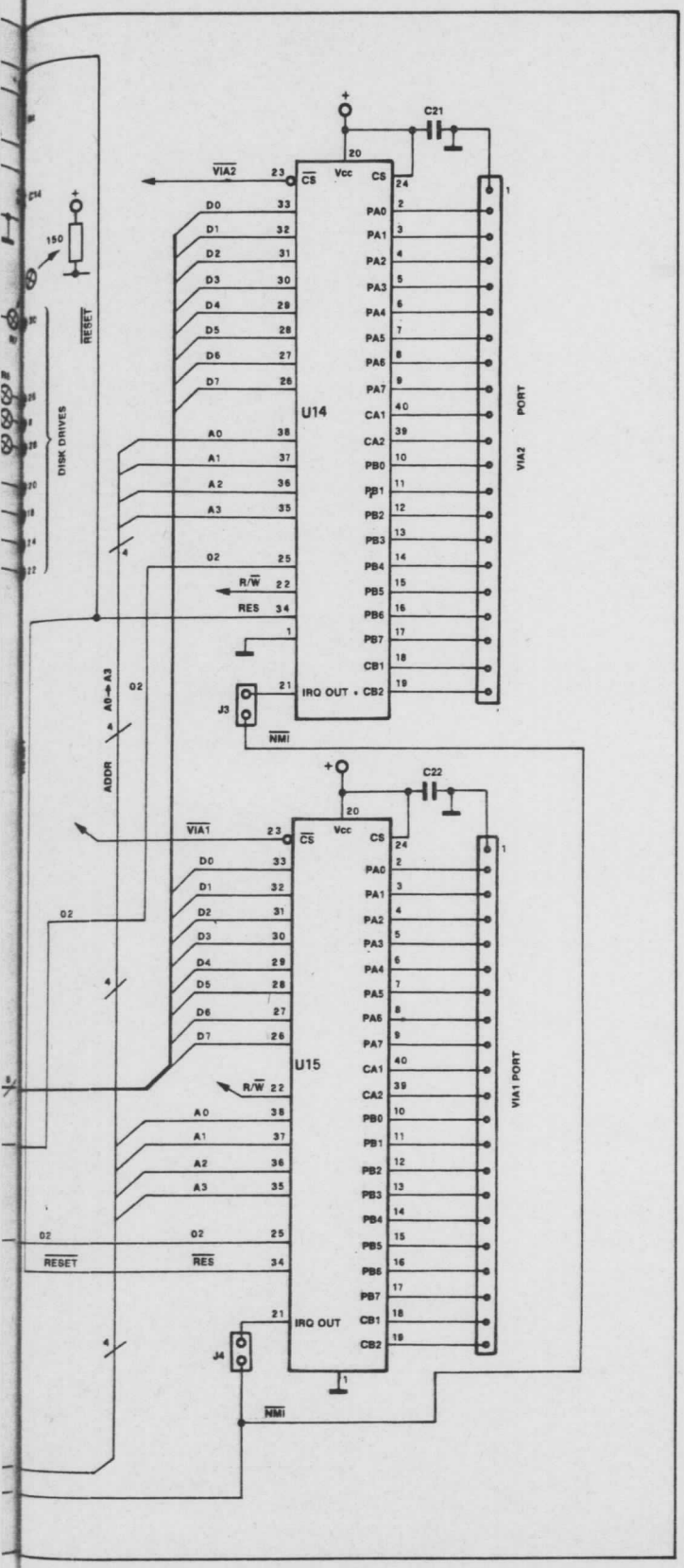


Figura 2 - Schema elettrico della FORTH-Disk-Card.

FORTH Card



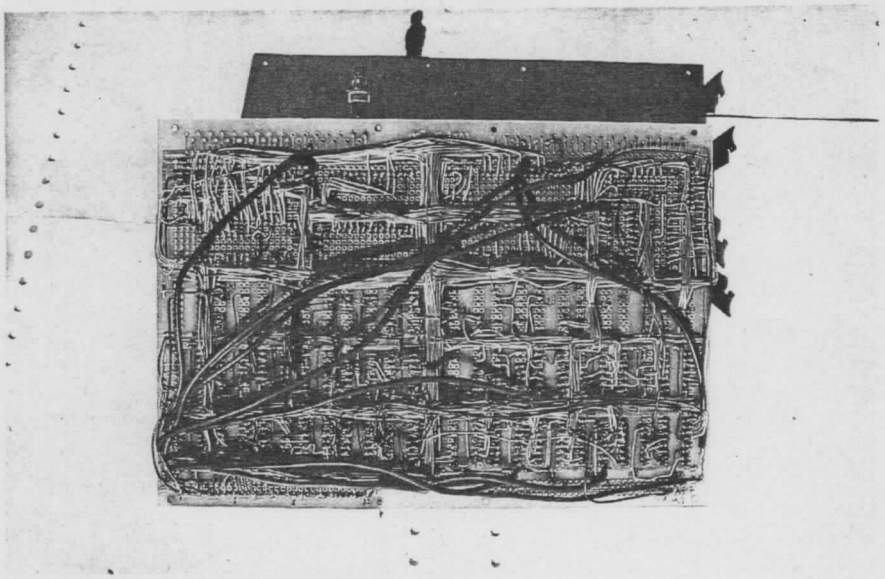


Foto 2 - L'intrico di cavi che effettuano le connessioni fra gli integrati del prototipo (non è stato un montaggio facile...).

di memoria della figura 3. Con U8, dunque, si dividono i 16 Kbyte accessibili in otto banchi da 2 Kbyte: quattro sono destinati fissi a formare il banco di 8 Kbyte valido da $2000 a $3FFF (segnale di selezione ROM-negato), and-ati, da U9; gli altri quattro segnali di selezione sono sia and-ati per formare un RAM-negato ampio 8 Kbyte, sia disponibili separatamente sul ponticello J19, in modo da poter essere dirottati su uno qualsiasi dei tre zoccoli bytewide. Questi ultimi sono chiamati U10, U11 ed U12, e diversi ponticelli permettono di adattare ogni zoccolo ad ospitare i più svariati tipi di memoria, RAM o EPROM o ROM, da 2 a 8 Kbyte. U10 è "più dotato" degli altri, disponendo infatti anche del ponticello J8, che permette di adottare memorie dinamiche bytewide, come la 2187 Intel. In tal caso c'è bisogno di un segnale di refresh (rinfresco periodico della memoria dinamica), che deve essere applicato sul piedino 1 di U10. L'impulso di refresh attiva, internamente alla memoria, una circuiteria che gestisce, in modo autonomo e trasparente, il mantenimento dei dati senza che siano necessari, all'esterno, contatori e multiplexer come per le normali memorie dinamiche da 64 Kbyte x 1. Gli integrati U5, U6 ed U18 manipolano il clock (Fase 2) e generano tutte le temporizzazioni necessarie per pilotare senza problemi il chip dinamico 8 Kbyte x 8. Tale soluzione è stata adottata a causa della perdurante carenza dei chip statici CMOS da 8 Kbyte x 8, che sono quasi introvabili e, per di più, enormemente cari (ma anche i chipponi dinamici non scherzano!).

Lo schema sarebbe tutto qui, se non fosse per U13, U14, U15 ed U16, che gestiscono l'interfacciamento con il bus Minimicro, grazie anche alla porta U17 che può disabilitare la decodifica interna. Ma tutto ciò necessita di una adeguata discussione a parte. Intanto, nella fotografia 3 potete riconoscere un particolare del prototipo V1.0, che mostra "il cuore" del sistema: il single chip, la RAM da 8 Kbyte, la ROM di sviluppo da 8 Kbyte ed il floppy disk controller.

La FORTH Card ed il bus Minimicro

Il bus Minimicro è uno standard di connessione fra schede Eurocard, creato già da alcuni anni dalla EC di Brescia ed oramai largamente diffuso in Italia. Le schede Minimicro sono veri e propri componenti o, per meglio dire, dei "mattoni" già pronti all'uso che basta mettere assieme sul bus di un semplice motherboard per creare un'apparecchiatura a microprocessore tale da soddisfare tutte le esigenze di controllo in ambiente industriale. Noi stessi abbiamo eseguito svariati lavori con queste schede, che, a nostro giudizio, non hanno rivali nel campo delle applicazioni 6502 e 6809: per esempio, le Rockwell sono giunte dopo e costano quasi il doppio, per le stesse funzioni; le Motorola non sono Eurocard; le Thomson-Efcis, ed altre simili, hanno costi più alti e meno funzioni disponibili.

Detto questo, ci è sembrato logico rendere Minimicro-compatibile anche la scheda CPU-FORTH, dato che gli utenti potranno così beneficiare, se la useranno in applicazioni dedicate, di tutta una serie di schede applicative di espansione, quali RAM, ADC, I/O di potenza industria-

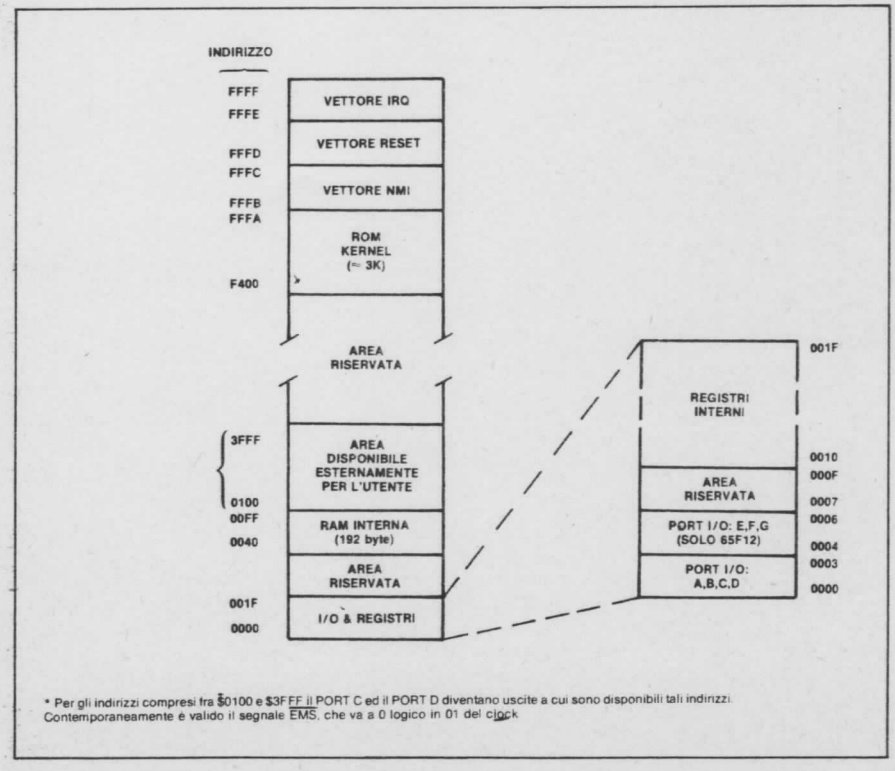


Figura 3 - La mappa della memoria indirizzabile dal 65F11.

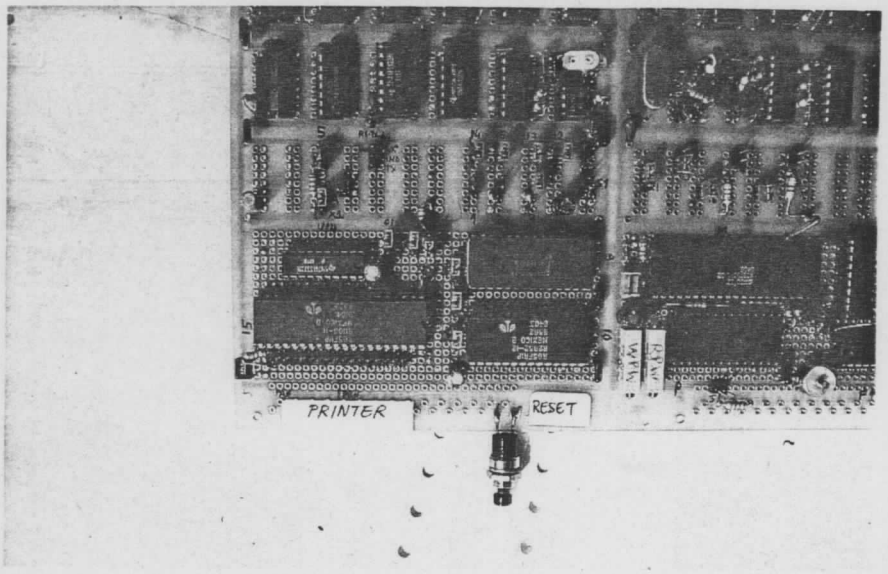


Foto 3 - Un particolare del prototipo: sono visibili i 4 chip più importanti (65F11, RAM, ROM, FDC).

le, floppy controller intelligenti, interfacce seriali e parallele, memorie video grafiche ed alfanumeriche sincrone, ecc. Si schiude così un mondo di applicazioni realizzabili con la massima facilità, grazie all'unione del FORTH con la versatilità delle funzioni disponibili su questo bus.
È solo dal punto di vista tecnico, però, che si può apprezzare la funzionalità dell'impostazione del bus. Il presupposto, infatti, è che la decodifica di sistema, ovvero quella implementata sulla sola scheda CPU (la FORTH Card, nel nostro caso), non deve limitare la inserzione sul bus di tutte le possibili schede applicative. Si è realizzata così la capacità di disabilitare dall'esterno la decodifica di sistema, per cui è possibile sovrapporre funzioni esterne a quelle già esistenti sulla scheda CPU, senza che da questo derivi un conflitto sul bus. Vediamo un esempio.
Installiamo sul bus la sola FORTH Card con un chip di ROM (8 Kbyte) in U11 ed un chip di RAM (8 Kbyte) in U10. All'accensione il resistore R20 tiene a 1 logico la linea EA-negato (External Address System Decode Disable), per cui U8 funziona normalmente ed il sistema prende l'avvio come al solito. Adesso, giusto per il gusto di fare una prova, inseriamo sul bus una scheda di RAM da 8 Kbyte, la cui decodifica sia predisposta per l'indirizzo di base $0000. Per fare ciò usiamo un semplicissimo motherboard con i connettori femmina Eurocard collegati in parallelo pin-to-pin. Resettiamo il sistema (S1 di figura 1) e, che ci crediate o no, non cambia nulla rispetto a prima!
O meglio, tutto è cambiato, ma è andata così: dopo il Reset, tutte le operazioni di scrittura/lettura nella zona di RAM da $0100 a $1FFF generano, ancora in fase 1 del clock, l'indirizzo della locazione chiamata. Tale indirizzo è in ogni caso emesso sul bus da U14 e da U15, i quali vengono aperti dal segnale EMS, che, di conseguenza, ha anche la funzione di Valid Memory Address. Se sul bus c'è una scheda applicativa, tutti gli indirizzi generati giungono alla sua decodifica, che può essere fatta, ad esempio, come quella della figura 4. Se l'indirizzo è tale da far

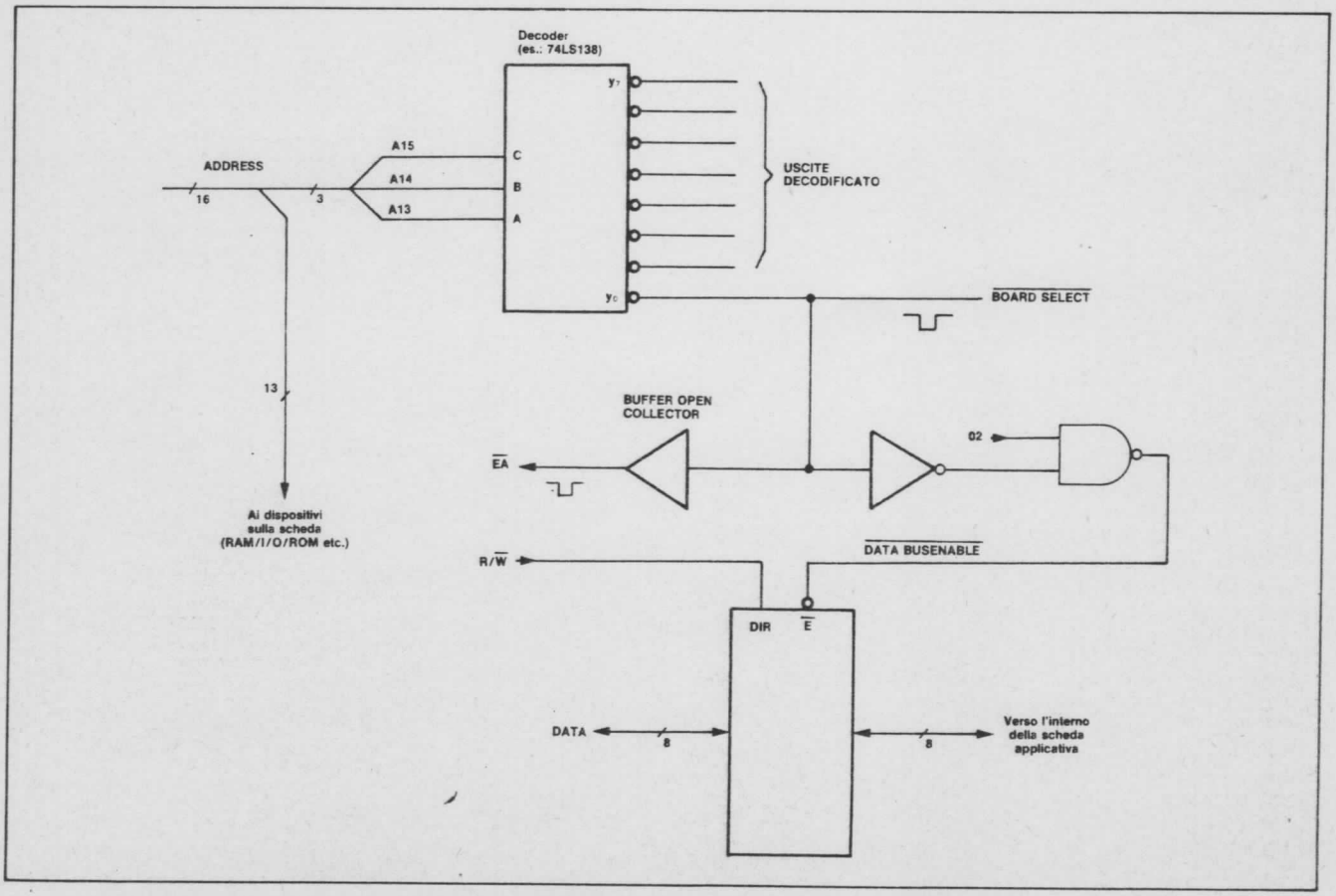


Figura 4 - Un esempio di decodifica di una scheda applicativa compatibile col bus Minimicro.

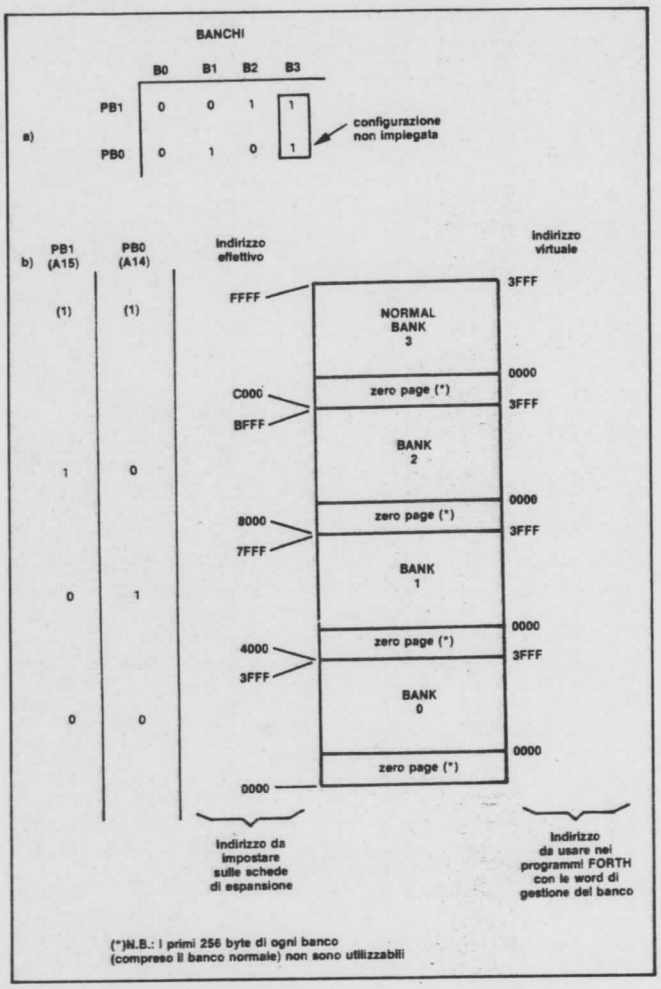


Figura 5 - I quattro banchi di memoria gestibili dal 65F11.

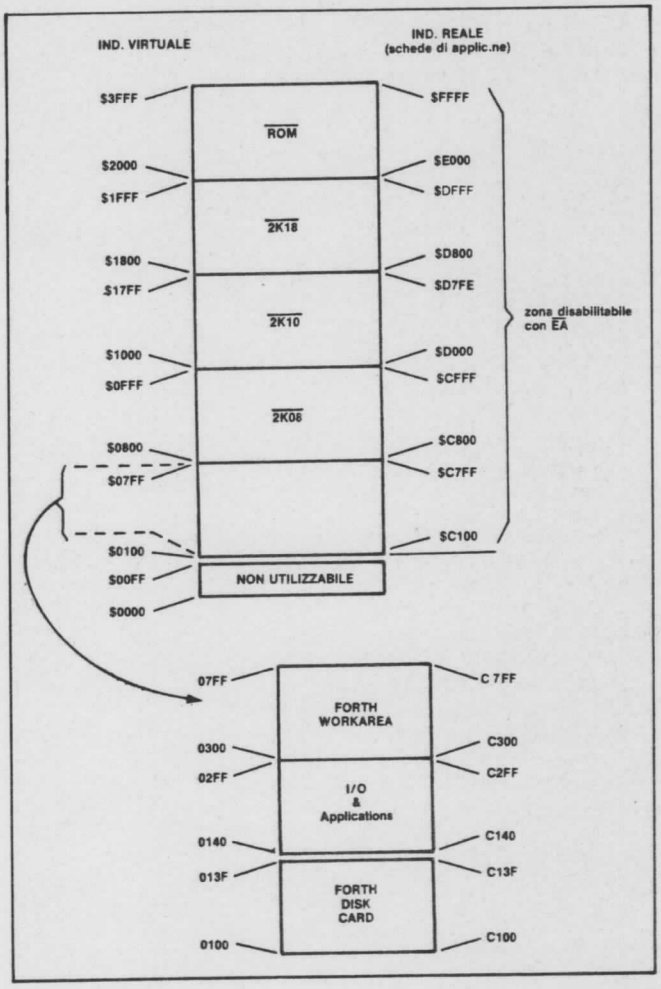


Figura 6 - Il banco normale (3) e la sua suddivisione, con i segnali di selezione disponibili.

produrre il segnale di Board-select-negato, questo stesso segnale, bufferato con un open-collector, diviene il famoso "External Address negato": in pratica, la scheda ha riconosciuto un indirizzo valido nel campo di preselezione della sua decodifica e ha segnalato questo riconoscimento alla CPU, abbassando la linea EA-negato. A questo punto (si torni alla figura 1) la scheda CPU ha la sua decodifica disabilitata, per cui nessun chip select viene generato on-board e nessun dispositivo on-board (RAM, ROM, I/O) può rispondere. È chiaro adesso che una scheda applicativa esterna ha sempre la priorità rispetto alla mappa di memoria della scheda CPU e sono possibili sovrapposizioni senza alcuna interferenza sul bus. Si noti ancora dalla figura 1 che la decodifica di sistema è abilitata solo in fase 2, e quindi ciò impedisce in modo assoluto ogni conflitto, dato che all'inizio della fase 2 una scheda esterna, se chiamata, ha già sicuramente risposto.

Il resto del ciclo, una volta che la decodifica di sistema sia stata spenta, si svolge come segue: all'inizio della fase 2 i buffer dei dati, della scheda applicativa e della scheda CPU si attivano nello stesso senso, ed il dato può passare verso la scheda applicativa (scrittura) o verso la scheda CPU (lettura). La fase 2 si conclude con la disattivazione dei buffer, con la rimozione degli indirizzi da parte della CPU e la conseguente disattivazione della linea EA-negato. Il bus è pronto così a gestire il ciclo seguente.
Ultima nota: è lampante che il trasferimento dei dati avviene solo e solamente nella fase 2 del clock di sistema, mentre la fase 1 è destinata alla stabilizzazione degli indirizzi. È allora possibile, con questo bus, implementare funzioni applicative, come memorie video, schede grafiche ad alta risoluzione, e via dicendo, il cui funzionamento sia sincrono rispetto al clock e trasparente (cioè invisibile) nei confronti del normale funzionamento della CPU: sono in tal modo eliminati disturbi (flicker) in tutte le applicazioni video e non sono necessari tempi morti di attesa (esempio del ritorno di riga o di quadro). Nel prossimo articolo vedremo proprio una applicazione con una scheda video alfanumerica mappata sul bus.

La gestione della memoria a segmenti

La discussione del paragrafo precedente è stata impostata tenendo conto di una normale "finestra" di memoria indirizzabile di 16 Kbyte, precisamente da $0000 a $3FFF. Infatti, il 65F11 genera esternamente solo 14 indirizzi, da A0 ad A13, e quindi tutti gli indirizzi da $4000 a $FFFF "girano" solo all'interno del chip. La cosa è facilmente verificabile se, una volta acceso il sistema, si comanda per esempio:

HEX FB4D FF DUMP <cr>

Sul video comincerà ad apparire il dump della ROM interna di kernel a partire dall'indirizzo di Reset:

A2 FF 9A D8 18 A9 E0

Quindi tutti e 16 gli indirizzi ci sono, ma solo i primi 16 Kbyte di memoria sono utilizzabili esternamente al single-chip, dai quali poi va tolta la pagina 0, che nel 65F11 vede mappati gli I/O, la RAM interna e lo stack, come è sempre dettagliato in figura 3. Ci possiamo allora domandare, giustamente, se 16 Kbyte disponibili all'esterno siano sufficienti per un utente FORTH. La risposta è senz'altro affermativa, soprattutto per le ragioni viste nello scorso capitolo quando abbiamo analizzato la compattezza dei programmi compilati in FORTH. Con 16 Kbyte, infatti, potrete usare gli ultimi otto per il vostro programma in EPROM (al posto della ROM di sviluppo); la rimanente area di memoria può invece essere impegnata da RAM, la cui estensione andrà da un minimo di 2 Kbyte ad un massimo di 8 Kbyte (6 Kbyte almeno per poter usare i dischi). Ricordando che il rapporto di "compattezza" fra codice macchina nativo e codice compilato FORTH è almeno di 2.5 : 1, se ne deduce che 8 Kbyte di ROM sono giganteschi per una applicazione in FORTH!
D'altronde, vi sono spesso dei casi in cui è necessario impegnare un'area considerevolmente cospicua di memoria per applicazioni fra le quali la più tipica è la presentazione su display video di testi alfanumerici o di grafici in media/alta risoluzione. In tal caso è chiaro che una memoria video bit-mapped con una estensione, per esempio, di 16 Kbyte non può essere inserita sul bus come una qualunque scheda applicativa. Fortunatamente, il problema si risolve sfruttando la eccezionale versatilità del 65F11, che può indirizzare una memoria esterna suddivisa in segmenti fino alla incredibile estensione di 4 Mbyte. Per comprendere come è gestito tale indirizzamento occorre osservare la figura 5.

La parte a) mostra il "trucco": le linee di indirizzo più significative (da A14 in su) sono realizzate con le uscite PB0, PB1, PB2, ... , PB7 del Port B dello stesso chip 65F11. Per comodità, limitiamo la discussione alle sole linee A14 ed A15, gestite da PB0 e da PB1: è una limitazione solo parziale, dato che, in ogni caso, le schede applicative esterne hanno decodifiche progettate per un massimo di 16 linee di indirizzo, per cui un numero maggiore non servirebbe a nulla (ma voi, espandendo la decodifica, potrete utilizzare tutti i 4 Mbyte).
Durante un normale funzionamento il Port B non è toccato e le sue linee sono ingressi tenuti ad 1 logico dai pull-up R45 ed R46 (figura 1). Per il FORTH l'indirizzo "virtuale" va da $0000 a $3FFF, ma in realtà l'indirizzo effettivo, rapportato a 64 Kbyte aggiungendo A14 (PB0) ed A15 (PB1), va da $C000 a $FFFF. In questo banco, che è dunque la normale area di lavoro del FORTH, tutte le schede applicative esterne alla scheda CPU dovranno essere fisicamente predisposte per rispondere ad indirizzi fra $C000 e $FFFF, ma saranno chiamate nei programmi con gli indirizzi virtuali da $0000 a $3FFF. In questo banco non è inoltre necessaria alcuna procedura di gestione della memoria segmentata. Se diamo il nome al banco, leggendo il valore di A14 ed A15, questo sarà dunque il "Banco 3".
Il Banco 2, invece, è richiamato quando A14 ed A15 valgono, rispettivamente, 0 ed 1, ed analogo discorso vale per i banchi 1 e 0. I due indirizzi vengono settati durante la esecuzione di quattro speciali word del FORTH, che eseguono dei caricamenti (load) o delle memorizzazioni (store) relative al banco richiamato. Per esempio, se noi abbiamo una memoria video ampia 2 Kbyte, che ci serve come display alfanumerico, potremmo inserirla nel Banco 2, predisponendo la sua decodifica per l'indirizzo-base $8000. Mentre si lavora nel banco 3 la scheda è inerte (A14 e A15 sono diversi). Quando invece vogliamo, modificare il contenuto di una sua cella, eseguiamo in FORTH:

HEX 55 014A 2 BANKC!

e, come per magia, la prima cella della memoria assumerà il valore $55. Questo accade perché il FORTH riconosce la word BANKC! con i tre argomenti sullo stack, che sono: numero da scrivere, indirizzo virtuale (cioè, come sempre, riferito alla sola finestra di 16 Kbyte) e numero del banco da gestire. Durante la esecuzione della word il 65F11 predisporrà opportunamente PB0 e PB1, e quando verrà effettuata la scrittura la scheda potrà rispondere come se fosse normalmente nel Banco 3. Dopo la esecuzione della word, PB0 e PB1 vengono riportati allo stato di ingressi a 1 logico ed il processo torna nel Banco 3. In pratica, la BANKC! è l'analoga della C! (store a 8 bit), ma con la gestione automatica del segmento di 16 Kbyte scelto; naturalmente, esiste anche la BANKC@ (fecch, ovvero load a 8 bit) che ha comportamento analogo, con sintassi: iiii bn BANKC@ (iiii = indirizzo da leggere, bn = numero del banco).
Con la FORTH Card i banchi validi sono 0, 1 e 2, mentre chiamare il Banco 3 è perfettamente inutile, dato che la normale finestra di lavoro è già il Banco 3.
Vi sono altre due word che gestiscono i segmenti della memoria esterna: sono word di utility, in quanto la prima, BANKEXECUTE , permette di testare una word trasportata in RAM o ROM fuori dal ban-

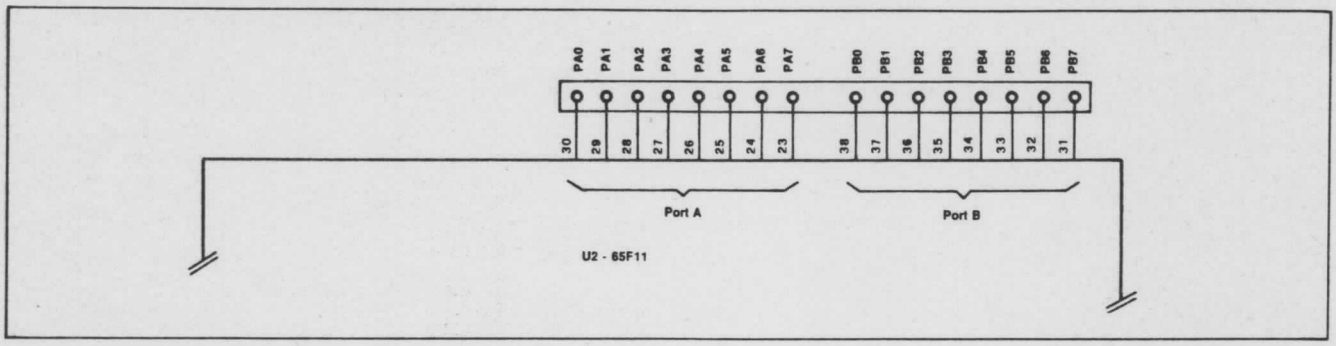


Figura 7 - Disposizione e piedinatura delle 16 linee dei Port A e B del 65F11.

co normale (il 3, come si è detto); la BANKEEC! gestisce invece la programmazione di una EPROM posta in un banco esterno.

Mappa di memoria nel Banco 3 (normale)

La figura 6 mostra la suddivisione del Banco normale (il Banco 3) con i segnali di selezione (Chip Select) disponibili dalla decodifica on-board della FORTH Card. Il segnale EMS-negato è valido da $0100 a $3FFF (virtuali), ovvero da $C100 a $FFFF (effettivi). Ogni scheda di espansione compatibile col bus Minimicro, che verrà inserita sul motherboard assieme alla FORTH Card, dovrà avere la decodifica impostata fra $C100 e $FFFF, per rispondere nel banco normale. Come noto, da $C000 a $C0FF effettivi gli indirizzi esterni non sono validi (ed il segnale EMS-negato non commuta). Consigliamo gli utilizzatori della FORTH Card di organizzare le schede applicative come segue:

- da $C100 a $C13F scheda FORTH-Disk-Card (vedi dopo);
- da $C140 a $C2FF schede I/O seriali, parallele, ADC, ecc.;
- da $C300 in su RAM dedicata al FORTH (buffer, vocabolario, ecc.);
- da $2000 a $3FFF ROM di sviluppo o col proprio programma.

Il FORTH del 65F11 non tocca la RAM fra $C100 e $C2FF, che è così tutta disponibile, salvo i 64 byte iniziali dedicati alla eventuale inserzione sul bus della scheda FORTH-Disk-Card (che è necessaria solo usando la FORTH Card come sistema di sviluppo FORTH). In ogni caso, si deve ricordare che i primi 256 byte di ogni banco non sono utilizzabili: per sicurezza il 65F11 non emette il segnale EMS.

Selezione dei ponticelli sulla FORTH Card

Vi sono ben 21 ponticelli sulla FORTH Card che permettono di adattare certe sue caratteristiche in modo flessibile. J1 è normalmente aperto e abilita il divisore per 2 del clock, interno al 65F11. J2 porta + 5 o + 12 V al piedino 14 di U4, in modo che si possa scegliere un 74LS00 o un buffer RS-232C del tipo 1488, con cui 1488 vanno chiusi i contatti 1 e 2. J3 ha la medesima funzione coi -12 V. Col 1488 vanno collegati 2 e 3. J4, J5, J6 e J7 sono normalmente lasciati aperti, se U5 ed U6 sono 74123; vanno chiusi, se si usano 74LS123. J8 porta il segnale di refresh alla eventuale memoria dinamica inserita in U10, e J9 porta il Write-Enable ricostruito (2 & 4) oppure il normale WE-negato (2 & 3) oppure +5V (2 & 1), in modo da adattare lo zoccolo ai vari tipi di memoria. Analoghe funzioni hanno i ponticelli da J9 a J14, i cui segnali sono chiaramente indicati sullo schema di figura 1 e per la cui impostazione occorrerà di volta in volta riferirsi alle note applicative della memoria (RAM/ROM/EPROM o EEPROM) impiegata sul rispettivo zoccolo. Infine i ponticelli da J15 a J18 sono normalmente chiusi, per realizzare il segnale RAM-negato di 8 Kbyte, ma l'utente può aprire dei buchi in questo segnale di selezione ed utilizzare il ponticello J19 per indirizzare i vari segnali disponibili ai tre zoccoli di memoria. E, non dimentichiamolo, in ogni caso la mappa di memoria è alterabile a piacere, inserendo semplicemente schede sul mother, grazie all'External Address ed alla decodifica disabilitabile.
La figura 7, infine, mostra la piedinatura dei due port del 65F11, che apparivano in figura 1 solo con un asterisco sul fianco di U2, per manifesta mancanza di spazio! Ricordiamo che PB0 e PB1 sono usati come A14 ed A15 per la gestione dei segmenti di memoria, per cui è sensato disabilitare queste due linee quando si adopera il Port B come normale I/O parallelo: questo spiega la presenza dei due ponticelli J20 e J21, che vanno lasciati aperti se si vogliono tenere A14 ed A15 stabili a 1 logico (per esempio: quando si controlla una stampante con il Port B ed il Port A, come vedremo più avanti).

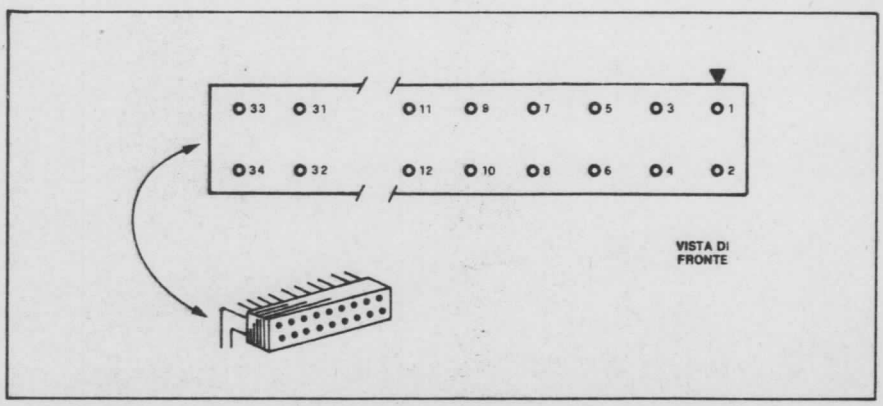


Figura 8 - Vista frontale del classico connettore a 34 poli per cavo piatto, che si usa per collegare la FORTH-Disk-Card ai drive.

Caratteristiche della FORTH-Disk-Card

Con la FORTH-Disk-Card inserita sul motherboard assieme alla FORTH Card basta aggiungere un comunissimo ter-

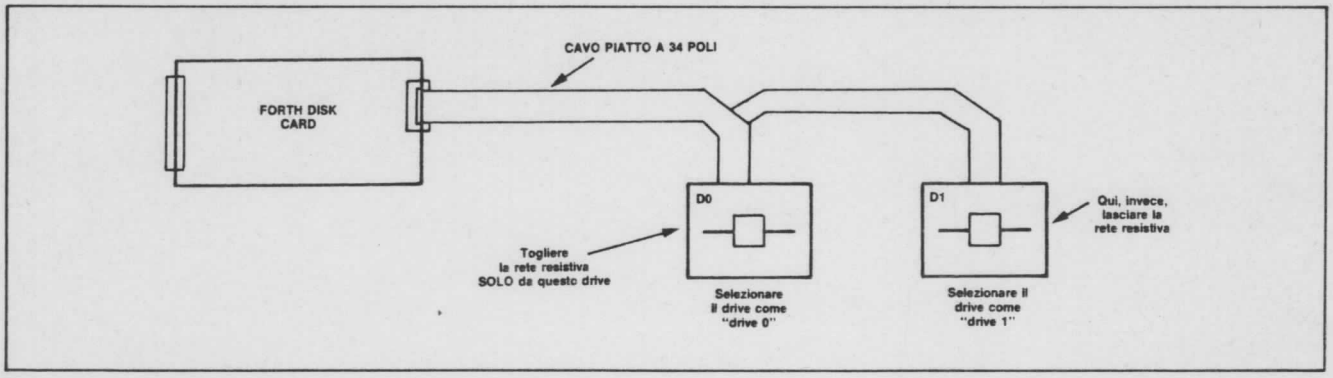


Figura 9 - Collegamento FORTH-Disk-Card/drive. Si noti dove occorre lasciare la rete resistiva del drive: cioè solo sull'ultimo.

minale seriale e, tipicamente, due floppy disk drive per avere un potente sistema di sviluppo, senza dubbio di ridotte dimensioni, ma capace di insegnarvi l'uso del FORTH senza restrizioni. Abbiamo progettato la scheda aggiuntiva FORTH-Disk-Card in modo che la sua realizzazione sia economica e facile: il kernel FORTH sul 65F11 nella FORTH Card contiene tutte le routine di gestione di un floppy controller della serie Western Digital, e per questo ci siamo orientati sul chip WD2793, un floppy controller che include sia la logica di precompensazione sia il separatore di dati e necessita di appena 2 MHz di clock per controllare drive da 5"1/4. Visto che la decodifica, seppur spartana, era ancora sotto-utilizzata, abbiamo aggiunto due chip di I/O molto versatili: sono due notissime VIA 6522, che quindi dotano il sistema base, così messo assieme, di ben 40 linee di I/O parallelo in più, di quattro timer a 16 bit e di due shift register a 8 bit. L'unica restrizione è che la decodifica è fissa e la scheda risponde solo ai seguenti indirizzi (reali):

$C100 registro di stato (in lettura) e di comando (scrittura) del floppy controller;
$C101 registro di traccia;
$C102 registro di settore;
$C103 registro dati;
$C106 registro di stato (in lettura) e di controllo (scrittura), realizzati con U11 ed U8;
da $C110 a $C11F risponde la VIA 1;
da $C120 a $C12F risponde la VIA 2;
da $C130 a $C13F è uno spazio non usato.

Se la FORTH-Disk-Card è inserita in un motherboard assieme alla FORTH Card, ogni indirizzo effettivo fra quelli descritti, ovvero ogni indirizzo virtuale da $0100 a $013F nel Banco 3, farà rispondere la scheda.

Lo schema elettrico della FORTH-Disk-Card

La figura 2 mostra la circuiteria della FORTH-Disk-Card: è subito evidente la presenza di U4, che gestisce in pratica tutto il dialogo coi drive da 5"1/4, e di U14 ed U15, due VIA 6522 aggiunte per rendere più flessibile la scheda. La decodifica on-board risiede tutta in U1, U13 ed U2. U2 ha il compito di suddividere in quattro segmenti di 16 byte lo spazio di 64 byte del Board-Select. Tornando ad U4, si notano alcuni circuiti elementari di supporto, necessari per un sicuro funzionamento in un bus che non usa il DMA (Direct Memory Access) come il nostro: U11 è un finto registro di stato, che sostituisce quello interno al 2793. Leggendo U11 si riconosce lo stato di esecuzione/attesa/termine del comando impostato nel floppy controller, e si conosce anche se il motore gira U11 può solo essere letto, mentre U8, un latch ottale, può solo essere scritto ed invia ai drive i segnali di selezione e di controllo. Le connessioni al classico connettore da 34 poli, per cavo piatto, sono rappresentate dai pallini neri; tutti i piedini dispari del connettore (non appaiono nello schema) sono invece a massa. La disposizione è illustrata nella figura 9. Si noti che le linee che vanno ai drive sono open collector: è quindi necessario che l'ultimo dei drive collegati alla FORTH-Disk-Card possegga una terminazione resistiva da 150 ohm (vedi figura 9).
Si noti anche che in tutti gli schemi le

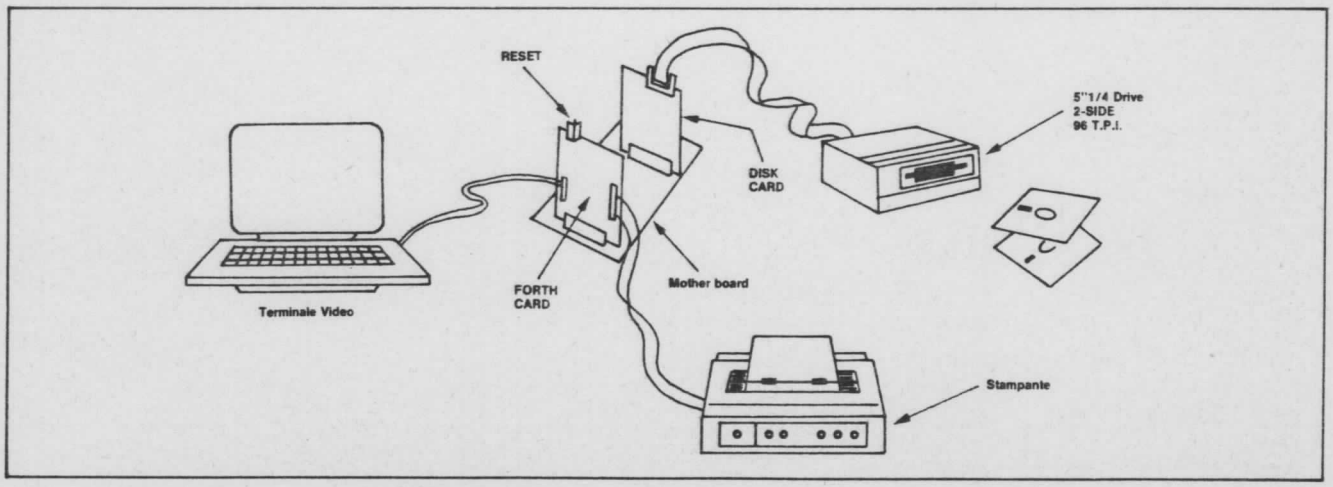


Figura 10 - Il sistema FORTH Card completo di terminale, disco, stampante.

FORTH Card

PICCOLO GLOSSARIO

Bytewide: è uno standard nella disposizione dei piedini per i chip RAM, ROM, EPROM più recenti. In tal modo, con solo pochissime modifiche (bastano di solito un paio di ponticelli di selezione), è possibile usare uno stesso zoccolo da 28 piedini per moltissimi tipi di integrato.

RS-232C: è uno standard per i livelli di segnale usati in una tramissione seriale. I segnali vanno infatti da − 12 a + 12 volt.

Baud: misura della velocità di trasmissione seriale. Corrisponde a bit/sec.

Full-Duplex: si dice di uno scambio di dati tramite una linea seriale, in cui il canale di trasmissione è del tutto indipendente da quello di ricezione.

Decoder 3→8: è un dispositivo, di solito integrato in un solo chip poco costoso, che ha un certo numero di uscite pari, al massimo, a 2 elevato al numero degli ingressi decodificati. Dunque con 3 ingressi si hanno otto uscite. Una ed una sola delle otto uscite è attivata in corrispondenza di una ed una sola combinazione degli ingressi. È così possibile riconoscere una combinazione in ingresso (tipicamente degli indirizzi) ed emettere un corrispondente segnale di selezione.

Memoria (RAM) dinamica: è una memoria leggibile e scrivibile, la quale necessita di una cura periodica particolare, che consiste nella applicazione di un opportuno segnale di refresh.

Memoria (RAM) statica: è una memoria leggibile e scrivibile, che non ha bisogno di alcuna sincronizzazione particolare per ritenere i dati immagazzinati, al contrario, dunque, di una RAM dinamica.

Refresh: segnale periodico che occorre applicare ad una RAM dinamica perché non perda i dati immagazzinati. Significa "rinfresco" e consiste nel forzare una rilettura dei dati, che a sua volta, all'interno del chip, forza una riscrittura.

Motherboard: circuito stampato che supporta dei connettori femmina, collegati in parallelo, su cui si innestano le schede che rispettano la medesima disposizione delle linee di bus.

Floppy Disk Controller: circuito integrato molto complesso che gestisce, semplificandole, tutte le operazioni di scrittura e lettura di dati su un dischetto flessibile (floppy disk).

Precompensazione: operazione durante la quale il Floppy Disk Controller altera opportunamente i dati che vanno scritti sul disco, accelerandoli, affinché durante una successiva rilettura sia minore la probabilità di errori.

Separazione: operazione, eseguita dal Floppy Disk Controller, che consiste nel decifrare gli impulsi elettrici letti sul disco magnetico dalla testina, in modo da ricreare i dati originari.

DMA: Direct Memory Access, cioè l'azione che una periferica compie quando chiede alla CPU di arrestarsi un istante, in modo da poter riempire la memoria con dei dati che arrivano molto velocemente (esempio: da un disco o da una linea seriale). Significa "accesso diretto in memoria (senza l'ausilio della CPU)".

terminazioni a forma di paletta tratteggiata indicano i punti del connettore del bus Minimicro, secondo la descrizione della Tabella 1. Per nostra comodità e per risparmiare circuiti logici, abbiamo sfruttato le linee ammesse come "user" per inviare alla FORTH-Disk-Card i segnali: A7-negato, EMS-negato, OE-negato ed il

```
1 LIST
SCR # 1
  0 ( CENTRONICS DRIVER FOR COMPUTERJOB-ELETTRONICA FORTHCARD)
  1 ( OUTPUT TO VIDEO ALSO) BASE @ HEX
  2 LATEST DP @ 100 + H/C
  3 CODE STB-PRT 10 3 RMB. 0 2 RMB. 0 2 SMB. RTS. END-CODE
  4
  5 CODE ACK-WT BEGIN. 11 3 BITSET UNTIL. RTS. END-CODE
  6
  7 CODE HOUT 1 STA. D # CMP. 0= NOT IF. ' STB-PRT CFA @ JSR.
  8 ' ACK-WT CFA @ JSR. THEN. F5EF JSR. RTS. END-CODE
  9 0 HEADERLESS !
 10 : P-ON [ ' HOUT CFA @ ] LITERAL 0046 ! ;
 11 : P-OFF F5EF 0046 ! ;
 12 : FORM-FEED 0C EMIT ;
 13 ' P-ON LFA ! BASE !
 14 DECIMAL
 15 ;S
OK
```

Tabella 1 - Il list dello schermo 1, con le word per la gestione della stampante.

clock del floppy controller (FCK, 2 MHz). Sempre negli schemi, ogniqualvolta si trova un cerchietto inquadrettato (esempio: TP1, J1, VIA1 PORT, ecc.), significa che vanno usati dei pin di tipo Molex, che possono facilmente essere wrappati, saldati, o usati come connettori ad inserzione, impiegando le rispettive femmine.

La taratura della FORTH-Disk-Card

Pur con ampi margini di tolleranza (ricordiamo che stiamo lavorando con i floppy da 5"1/4, e non con gli 8"), il WD2793 necessita di una taratura. Questa coinvolge la regolazione dei trimmer multigiri P1, P2 e del compensatore C18.

Per le operazioni di taratura è necessario un buon oscilloscopio e, naturalmente, una FORTH Card funzionante! Si inseriscono allora FORTH Card e FORTH-Disk-Card su un motherboard, senza collegare i drive. Si applica l'alimentazione e si controlla che il segnale FCK sia un'onda quadra di 2 MHz. Adesso si deve agire sul ponti-

cello TEST (J2), che normalmente è aperto, secondo la seguente sequenza:
1) con J2 aperto si deve resettare il sistema (S1 sulla FORTH Card);
2) si chiude J2;
3) si osserva il segnale al punto di test TP1: è un'onda quadra;
4) si agisce sul potenziometro P2 per il duty factor desiderato, che definisce il valore di precompensazione adottato per i drive (si veda il manuale dei drive in possesso);
5) si riapre J2.
C'è poi da regolare il separatore di dati, con queste operazioni in sequenza:
1) con J2 aperto si resetta il sistema;
2) si chiude J2;
3) si osserva il segnale su TP2: deve essere un'onda quadra con un periodo intorno ai 500 ns;
4) si regola il potenziometro P1 per ottenere un periodo effettivamente di 500 ns;
5) si osserva il segnale su TP3: anche qui è un'onda quadra;
6) si regola il compensatore C18 per una frequenza di 250 kHz;
7) si riapre J2.
Ecco le funzioni degli altri ponticelli presenti sullo schema della FORTH-Disk-Card: oltre a J2, c'è J1, che normalmente si lascia aperto, ma che permette di abilitare o disabilitare la precompensazione su tutte le tracce, o di abilitarla solo dalla traccia 43 in poi. J3 e J4 permettono di collegare alla linea NMI-negato di sistema le uscite di interrupt delle due VIA 6522: ricordo che il 65F11 non ha ingresso di IRQ (che è interno al chip), ma solo di NMI; IRQ è l'interrupt mascherabile, mentre l'NMI è quello non mascherabile.

LINEE DEL BUS MINIMICRO (CONNETTORE EUROCARD 64 POLI 32+32 a & c)

a1 GND	c1 GND
a2 +12V	c2 −12V
a3 ...	c3 ...
a4 RESET-negato	c4 ...
a5 ...	c5 ...
a6 FASE 2	c6 ...
a7 FASE 2-negato	c7 ...
a8 RAM R/W (WE-negato)	c8 R/W
a9 user	c9 user
a10 A0	c10 D0
a11 A1	c11 D1
a12 A2	c12 D2
a13 A3	c13 D3
a14 A4	c14 D4
a15 A5	c15 D5
a16 A6	c16 D6
a17 A7	c17 D7
a18 A8	c18 user
a19 A9	c19 user
a20 A10	c20 user
a21 A11	c21 user
a22 A12	c22 ...
a23 A13	c23 ...
a23 A14	c24 ...
a25 A15	c25 ...
a26 NMI-negato	c26 ...
a27 ...	c27 ...
a28 ...	c28 ...
a29 ...	c29 ...
a30 EA-negato	c30 +5V STBY
a31 ...	c31 ...
a32 +5V	c32 +5V

Nota: i contatti segnati con "..." riguardano segnali che non servono con la scheda FORTH Card, in quanto previsti solo con CPU 6809 o 6502.

Tabella 2 - Disposizione delle linee nel bus Minimicro.

Introduzione all'uso della FORTH Card: una panoramica sul modo di lavorare in ambiente FORTH

Finalmente abbiamo un sistema FORTH Card completo, che probabilmente rassomiglierà al disegno della figura 10: un motherboard con infilate la FORTH Card e la FORTH-Disk-Card, almeno un drive a doppia faccia collegato a quest'ultima, un terminale video collegato alla FORTH Card e predisposto per 1.200 baud, 7 bit di dato, nessuna parità ed infine una stampante con interfaccia parallela tipo Centronics, collegata alla FORTH Card come è descritto nella tabella 4. Siamo quindi pronti per iniziare una breve panoramica delle funzioni della FORTH Card, il che servirà a renderci conto di come ci si muove in ambiente FORTH.
Accendiamo il sistema; sul video compare:
RSC FORTH V1.7
(è già qualcosa: vuol dire che funziona)
Quindi battiamo:
EMPTY-BUFFERS <cr>
ed il sistema risponde:
OK
Adesso abbiamo i dischi in linea, ma naturalmente il dischetto inserito nel drive non può funzionare, poiché non è formattato. Allora battiamo:
80 0 FORMAT <cr>
che significa: "formatta 80 tracce (per lato) del disco che è nel drive 0". Se tutto è ok, il drive inizia a frullare, viene eventualmente eseguito un restore e quindi la testina inizia ad avanzare ritmicamente, segnalando che, ogni volta, una traccia è stata generata sul disco. Adesso il disco è leggibile e scrivibile, anche se tuttora vuoto: in pratica abbiamo 640 Kbyte in linea, gestiti come memoria virtuale. Iniziamo quindi a lavorare in questa benedetta memoria virtuale, creando il primo "schermo", ove, per nostra futura comodità, scriveremo proprio la word che controlla la stampante, sfruttando i Port A e B del 65F11. Battiamo:

1 LIST <cr>
e sul video appare:

```
SCR # 1
 0
 1
 2
14
15
```

ove "SCR" sta per "screen", cioè schermo, e i numeri da 0 a 15 indicano le 16 righe di 64 caratteri ciascuna (vuote per ora) che compongono lo schermo in FORTH. Il FORTH dell'RSC 65F11 non possiede Editor, però è presente una utilissima word, la >LINE ("to-line"), che permette di immettere in memoria una riga qualsiasi dello schermo attualmente attivo. Per definizione ricordiamo che lo schermo "attivo" è l'ultimo per il quale era stato richiesto il LIST. Nel nostro caso, dunque, lo schermo attivo è il numero 1.
Se battiamo:
0 >LINE (CENTRONICS DRIVER FOR COMPUTERJOB-ELETTRONICA FORTH Card) <cr>
in pratica ordiniamo al FORTH di inserire al posto della vecchia riga 0 (dello schermo attualmente attivo) tutto l'argomen-

to a destra della word >LINE e che termina col <cr>. Il FORTH risponde OK, e se ridiamo: 1 LIST <cr>, rivedremo lo schermo numero 1, ma con la nuova riga 0. Ebbene, con la medesima procedura inseriamo le altre 15 righe definite nella Tabella 2. Se si commettono errori prima di battere il <cr>, è possibile correggerli con il tasto DELETE del terminale. Quando abbiamo terminato battiamo:
UPDATE FLUSH <cr> e quindi
1 LOAD <cr>
La prima operazione forza il salvataggio sul disco dello schermo appena generato; la seconda e ben più importante operazione mostra invece uno dei modi per compilare un programma in FORTH: ovverosia da disco. Gli altri due modi sono naturalmente il modo diretto (quando digitiamo una nuova definizione di word direttamente dalla tastiera del terminale) ed il modo SOURCE, in cui il programma da compilare arriva man mano dall'ingresso seriale (si dice che il programma è "downloaded" in formato testo, come accade sviluppando i programmi su un altro sistema e poi trasportandoli sulla FORTH Card).

Adesso abbiamo le word per controllare una stampante: P-ON la accende, P-OFF la spegne e FORM-FEED forza una nuova pagina. Perché non provarle subito? Dunque, battiamo:
P-ON <cr>
e quindi:
VLIST <cr>
per stampare tutte le word nel vocabolario Context/Current (quello FORTH), come è evidente dalla tabella 3. Nel caso volessimo vedere anche il vocabolario Assembler, dovevamo battere:
ASSEMBLER VLIST <cr>.
Ma questo non è ancora un uso in memoria virtuale: "memoria virtuale" significa infatti che l'utente non deve assolutamente preoccuparsi di gestire il caricamento/scaricamento sul disco dei file mentre li crea o li modifica. Per toccare con mano la gestione degli schermi in modo virtuale basta eseguire questa semplice sequenza:

```
P-OFF <cr>
1 LIST <cr>
0 > LINE ( SCREEN 1) <cr>
2 LIST <cr>
0 > LINE ( SCREEN 2) <cr>
3 LIST <cr>
0 > LINE ( SCREEN 3) <cr>
4 LIST <cr>
0 > LINE ( SCREEN 4) <cr>
P-ON 1 4 INDEX FORM-FEED P-OFF <cr>
```

Che cosa abbiamo fatto? Abbiamo modificato la prima riga degli schermi dall'1 al 4 ed abbiamo chiesto un "INDEX", ovvero una lista, del contenuto delle prime righe degli schermi dall'1 al 4. Provate e vedrete che tutti gli schermi sono stati aggiornati: eppure non abbiamo dovuto battere alcun comando di LOAD o di SAVE! Questo è, appunto, il metodo che il FORTH ha di gestire la memoria di massa in modo virtuale: l'utente lavora tranquillamente sul contenuto dei suoi schermi, anche saltando di qui e di là, senza preoccuparsi di salvare nulla, perché ci pensa automaticamente il Sistema Operativo, che conosce quando uno schermo è stato alterato rispetto all'originale, ed in tal caso provvede a riscrivere la nuova copia sopra la vecchia. Ogni schermo di

Tabella 3 - La stampa delle word contenute nel vocabolario FORTH del 65F11.

```
OK
VLIST
  46B FORM-FEED      450 P-OFF      439 P-ON       40B TASK
 3844 ADMP          3805 :DUMP     37CF FORMAT    367E FMTRK
 3674 BANKEXECUTE   3664 BANKEEC!  3657 BANKC@    364C BANKC!
 3641 EEC!          361E CASE:     35FD MEMTOP    35ED SCDR
 35DF SCSR          35D1 SCCR      35C3 MCR       35B6 IER
 35A9 IFR           359C PB        3590 PF        3584 PE
 3578 PD            356C PC        3560 PB        3554 PA
 3548 NMIVEC        3538 IRQVEC    3528 INTVEC    3518 INTFLG
 34EF C.CON         34AC .S        349F MON       345B VLIST
 33EC INDEX         33A0 LIST      3397 ?         3391 .
 338B .R            3384 D.        337D D.R       3375 #S
 336E #             3368 SIGN      335F #>        3358 <#
 3351 SPACES        333E WHILE     331A ELSE      3301 IF
 32E8 REPEAT        32CF AGAIN     32BF END       32A9 UNTIL
 3291 +LOOP         3279 LOOP      3264 DO        3257 THEN
 323A ENDIF         3226 BEGIN     3185 FORGET    3149 AUTOSTART
 3110 ?KERNEL       30BC HWORD     3086 H/C       306E '
 3068 SEEK          305F INIT      3056 DWRITE    304B DREAD
 3041 SELECT        3036 DISK      3023 R/W       3017 B/SCR
 3009 B/BUF         2FED -BCD      2FC9 -->       2F99 LOAD
 2F40 MESSAGE       2F0F )LINE     2EFB .LINE     2ED7 (LINE)
 2E94 DUMP          2E69 FLUSH     2E09 BLOCK     2DBF BUFFER
 2D9A EMPTY-BUFFERS  2D72 UPDATE    2D41 +BUF      2D38 M/MOD
 2D2E */            2D27 */MOD     2D1D MOD       2D15 /
 2D0F /MOD          2D06 *         2D00 M/        2CF9 M*
 2CF2 MAX           2CEA MIN       2CE2 DABS      2CD9 ABS
 2CD1 D+-           2CC9 +-        2CC2 S->D      2CB9 COLD
 2C4C ABORT         2C1D QUIT      2C0B (         2BF9 DEFINITIONS
 2BE1 ASSEMBLER     2BC9 FORTH     2B97 VOCABULARY 2B7D IMMEDIATE
 2B2D INTERPRET     2B02 ?STACK    2AE5 DLITERAL  2AB2 LITERAL
 2A94 [COMPILE]     29F1 CREATE    29C8 ID.       298B ERROR
 2977 (ABORT)       2949 -FIND     28F1 NUMBER    28E6 (NUMBER)
```

Seguito tabella 3.

```
2894 WORD          288B HOLD          2882 BLANKS        2877 ERASE
286D FILL          2846               2840 QUERY         2836 EXPECT
2804 ."            27FD (.")          27F4 -TRAILING     27E6 TYPE
27DD COUNT         27C1 DOES>         27AF (BUILDS       2795 ;CODE
277D (;CODE)       2771 DECIMAL       2765 HEX           2753 SMUDGE
273D ]             272D [             2715 COMPILE       26F8 ?CSP
26E4 ?PAIRS        26CC ?EXEC         26B3 ?COMP         269B ?ERROR
2686 !CSP          2670 PFAPTR        2656 NFA           2646 CFA
263C LFA           262A LATEST        2604 TRAVERSE      25F7 -DUP
25EE SPACE         25E4 PICK          25DB ROT           25D3 >
25CD (             25C7 U<            25C0 =             25BA -
25A8 C,            2595 ,             2587 ALLOT         2575 HERE
2560 ,/            2551 ALLOT/        253E HERE/         2524 DP/
251C 2-            2515 1-            250E 2+            2507 1+
2500 PAD           24F0 LIMIT         24DE FIRST         24D4 C/L
24C9 KHZ           24BE MODE          24B2 CSP           24A7 STATE
249A CURRENT       248B CONTEXT       247C SCR           2471 BLK
2466 PREV          245A USE           244F UABORT        243B VOC-LINK
242B HEADERLESS    2419 DP            240F FENCE         2402 WARNING
23F3 WIDTH         23E6 OFFSET        23D8 ULIMIT        23CA UFIRST
23BC B/SIDE        23AE CYLINDER      239E DISKNO        2393 HLD
238B DPL           2383 IN            237C CLD/WRM       2370 BASE
2367 UR/W          235E UPAD          2355 UC/L          234C R0
2345 S0            233E TIB           2336 BL            232F 4
2329 3             2323 2             231D 1             2317 0
2303 USER          22EC CODE          22D9 VARIABLE      22BE CONSTANT
22A3 :             2285 ;             226F C!            2268 !
2262 C@            225B @             2255 TOGGLE        224A +!
2243 BOUNDS        2238 2DUP          222F DUP           2227 SWAP
221E 2DROP         2214 DROP          220B OVER          2202 DNEGATE
21F6 NEGATE        21EB D+            21E4 +             21DE 0<
21D5 NOT           21CD 0=            21C6 R             21C0 R>
21B9 >R            21B2 LEAVE         21A8 ;S            21A1 RP@
2199 RP!           2191 SP!           2189 SP@           2181 XOR
2179 OR            2172 AND           216A U/            2163 U*
215C CMOVE         2145 FINIS         20F9 SOURCE        20E6 XOFF
20D5 XON           20CD CR            20C6 ?TERMINAL     20B8 KEY
20B0 EMIT          20A7 ENCLOSE       209B (FIND)        2090 DIGIT
2084 I             207E (DO)          2075 (+LOOP)       2069 (LOOP)
205E OBRANCH       2052 BRANCH        2047 EXECUTE       203B CLIT
2032 LIT           OK
FORM-FEED
```

64 x 16 = 1024 caratteri occupa 4 settori da 256 byte sul disco; ogni traccia di 16 settori contiene così quattro schermi ed il disco intero ne contiene 640, numerati da 0 a 639. Chiedete un 639 LIST e potrete nettamente avvertire il rumore della testina che si porta sull'ultima traccia (del lato 1 del disco).

stampante	65F11
D0	PB0
D1	PB1
D2	PB2
D3	PB3
D4	PB4
D5	PB5
D6	PB6
D7	PB7
Strobe	PA2
ACK	PA3

Tabella 4 - Collegamento fra FORTH Card e stampante parallela.

Da qui in poi, sfruttando la word " >LINE " ed usando la word " n LOAD " per scrivere un programma in uno schermo e farlo compilare, si può iniziare in modo sicuro l'avventura con il FORTH, imparando ogni word, testandola, inventando nuove word, fino ad arrivare alla stesura di un programma vero e proprio, in modo da sfruttare la FORTH Card in una applicazione dedicata. In questo caso si arriva finalmente a toccare con mano il passaggio fra il FORTH allo stato teorico e la sua applicazione pratica, quale può essere un controllo industriale e così via. Dovremo allora non solo sviluppare il programma e testarlo, ma dovremo anche realizzare una ROM o EPROM con il codice compilato, inserirla al posto giusto sulla FORTH Card e verificare che la scheda, ridotta a quel punto ad un microcomputer dedicato, si comporti secondo le aspettative.
Questi argomenti, assieme ad un "tutorial" più esteso sui rudimenti del FORTH e sull'uso dell'Assembler interno, saranno analizzati nel prossimo articolo.

A proposito della realizzazione

Usando il wire-wrap, o altri metodi consimili, potrete realizzare la FORTH Card senza eccessivi problemi (occorrono solo: tempo, pazienza e molta esperienza). Visti questi impegnativi presupposti, abbiamo realizzato per i lettori di **Bit** i circuiti stampati sia della FORTH Card che della FORTH-Disk-Card e, naturalmente, sono disponibili le schede già pronte all'uso. Per maggiori informazioni potete contattare direttamente l'autore (Ing. Paolo Bozzola - Via A. De Gasperi 13 - 25030 Roncadelle (BS) - Tel.: 030/278.278.3), che sarà a vostra completa disposizione per risolvere ogni ulteriore dubbio applicativo o hardware in relazione al presente progetto.
Infine desideriamo ringraziare per la cortesia e l'assistenza la Murata Erie S.r.l. di Milano, che ci ha fornito con la massima rapidità tutti i componenti ed i manuali Rockwell. ■

(Continua)

(La precedente puntata è apparsa sul n. 57).

Rom ¥.95'000 | F11 ¥. 37'000 (4 pins) ($1600)
F12 ¥. 48'000 (16 pins)

A LOW-COST DEVELOPMENT MODULE FOR THE R65F11 FORTH MICROCOMPUTER

by Joseph W. Harrce
Rockwell International, Semiconductor Products Division, Newport Beach, California

INTRODUCTION

The Rockwell R65F11 microcomputer implements a FORTH interpreter in a Rockwell R6500/11 single-chip microcomputer. With the addition of a Rockwell R65FR1 FORTH Development ROM the two-chip set is a complete FORTH development system. A module containing the R65F11, R65FR1, additional RAM memory, a terminal interface, and a floppy disk controller is all that is needed to provide a complete and inexpensive development and evaluation tool. This application note describes the design for such a module, complete with electrical schematic, printed circuit board layout artwork and parts list. In addition to hosting the R65F11 and R65FR1 devices the module contains an RS232 serial port, sockets for up to 16K bytes of RAM/ROM/PROM, parallel printer interface and a 5¼ in. floppy disk drive interface. The described development module can be easily fabricated using the artwork to build the printed circuit board.

In addition, artwork for a board to adapt an R65F12 into the development board is included. The three additional ports of the R65F12 are available via a connector on the adaptor board.

CIRCUIT DESCRIPTION

The R65F11 (U1) operates with a multiplexed address/data bus to access external memory. An external latch (U2) holds the address lines at the falling edge of EMS. The 555 timer (U5) generates a power-on-reset and the crystal oscillator circuit (U3) produces the master clock. Even though the R65F11 has an internal crystal oscillator, the 2 MHz signal generated by U3 is required by the floppy disk controller. A 256-by-8 bipolar PROM (U6) decodes the addresses and generates chip selects. The 74LS32 (U11) generates clocked chip select signals for the RAM devices. The use of the PROM facilitates changes in the memory map without changing the circuit. An example PROM pattern is given in Table 1 for a system consisting of 6K RAM (from $0200–$17FF) and 8K of ROM (from $2000–$3FFF).

16K: 8K RAM + 8K EPROM.

Each of the memory sockets can be configured for a number of different memory devices by changing the position of the various jumpers as shown in Table 2. Socket U7 can be used with an 8K × 8 static RAM, 2764-type or 2732-type PROMs. Socket U8 can be used with 8K × 8 static RAMS, 2K × 8 RAMs, 2764-type, 2732-type or 2716-type PROMs. Sockets U9 and U10 can use 2K × 8 RAMs, 2732-type or 2716-type PROMs. In addition, any socket may host a Rockwell R5213/2816 EEROM. The R65FR1 Development ROM is normally installed in U7.

The floppy disk controller (U12) is a Western Digital WD2793 device. The 74LS10 (U4) generates the $\overline{RD}$ and $\overline{WR}$ strobes while U14 and U18 buffer the input and output signals. U14 also forms a status register to allow the processor to read the FDC INT and DRQ lines. U15 forms an output latch to control the drive/side selection and the motor. A "test" jumper puts the WD2793 into the test mode for calibration.

A RS232 buffer is formed by the 1458 op-amp (U17) and a −5V supply is generated by U16. Connector J5 is configured to drive a parallel printer and J3 provides an expansion port for additional memory or I/O.

SOFTWARE CONSIDERATIONS

When the R65F11 is reset due to power up or reset, several variables assume default values which may need to be changed by the user. One of these is the top of memory pointer used by the disk interface. Reset assumes the presence of 8K of RAM from $0300–$1FFF. In the example given here, only 6K of memory is available ($0300-$17FF). In order for the disk buffers to work, the top of the memory must be set to 6K. The following commands will perform this function:

HEX 1800 MEMTOP DECIMAL [RETURN]

The R65F11 has been designed with vectored I/O so that user written I/O routines may be used as an alternative to the internal ones. The console input and output is vectored through the locations UKEY ($0044) and UEMIT ($0046) and the disk is vectored through UR/W ($30A). As an example of using these vectors, Listing 1 shows a method of patching UEMIT to direct the console output to a parallel printer port. This listing also shows a method of using the headerless code generation to create words and then forget the heads (only) of those words which will not be used again.

Listing 1. Implementing a Printer Driver

```
SCR #10
0   ( CENTRONICS DRIVER FOR RSC FORTH DEVELOPMENT
    BOARD )
1   ( FILTER OUT FORM FEEDS ) BASE @ HEX
2   LATEST DP @ 100 + H/C
3   CODE STB-PRT 10 3 RMB, 0 2 RMB, 0 2 SMB, RTS, END-CODE
4
5   CODE ACK-WT BEGIN, 11 3 BITSET UNTIL, RTS, END-CODE
6
7   CODE HOUT 1 STA, OA # CMP, O = NOT IF,  ' STB-PRT CFA
    @  JSR,
8    ' ACK-WT CFA @  JSR, THEN, RTS, END-CODE
9   0 HEADERLESS !
10  : P-ON [ ' HOUT CFA @ ] LITERAL 0046 ! ;
11
12  : P-OFF F5EF 0046 ! ;
13  : FORM-FEED OC EMIT ;
14  ' P-ON LFA ! BASE !
15  ;S
```

Document No. 29651N65

Application Note Order No. 2162
September, 1983

Table 1. Decode PROM (U6) Coding

The following configuration is assumed:

Funct	Address Space	Code	Signal	Socket
FDC	$0100–$013F	7F	FDC	U12
RAM1	$0200–$07FF	F7	MEM3	U10
RAM2	$0800–$0FFF	FB	MEM2	U9
RAM3	$1000–$17FF	FD	MEM1	U8
ROM	$2000–$3FFF	FE	MEMO	U7
Not Used	$1800–$1FFF	FF	No Chip Selected	
Page Zero	$0000–$00FF	FF	No Chip Selected	
Page One (ex FDC)	$0140–$01FF	FF	No Chip Selected	

msb/lsb	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	FF	FF	FF	FF	7F	FF	FF	FF	7F	7F	7F	7F	7F	7F	7F	7F
1	F7	F7	F7	F7	F7	F7	F7	F7	F7	F7	F7	F7	F7	F7	F7	F7
2	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB
3	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB
4	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD
5	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD
6	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
7	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
8	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE
9	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE
A	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE
B	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE
C	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE
D	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE
E	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE
F	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE

Table 2. Jumper Selection

Socket	Part Type	Jumper Position at Pin	
U7		**p27**	**p20**
	8K × 8 RAM	R/$\overline{W}$	$\overline{RMEMO}$
	2764 PROM	+5	$\overline{MEMO}$
	2732 PROM	+5	$\overline{MEMO}$
U8		**p27**	**p23** **p20**
	8K × 8 RAM	R/$\overline{W}$	A11 $\overline{RMEM1}$
	2K × 8 RAM	R/$\overline{W}$	R/W $\overline{RMEM1}$
	2764 PROM	+5	A11 $\overline{MEM1}$
	2732 PROM	+5	A11 $\overline{MEM1}$
	2716 PROM	+5	+5 $\overline{MEM1}$
U9		**p21**	**p18**
	2K × 8 RAM	R/$\overline{W}$	$\overline{RMEM2}$
	2732 PROM	A11	$\overline{MEM2}$
	2716 PROM	+5	$\overline{MEM2}$
U10		**p21**	**p18**
	2K × 8 RAM	R/$\overline{W}$	$\overline{RMEM3}$
	½ 2732 PROM	+5	$\overline{MEM3}$
	2716 PROM	+5	$\overline{MEM3}$

U1 R65F11

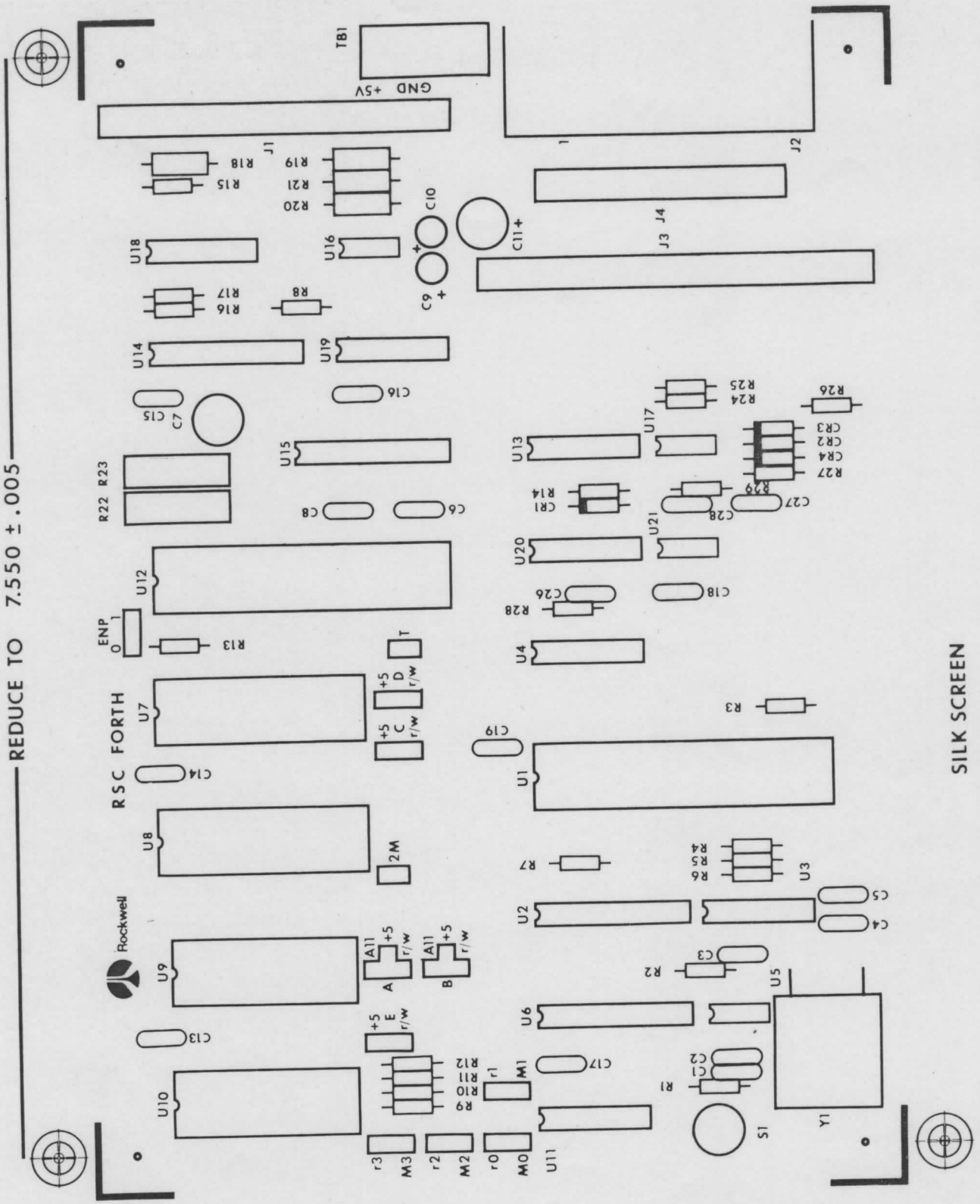
For 2 MHz device (R65F11AP), install jumper at Pin 1 to Ground; and set VDT for 2400 Baud.

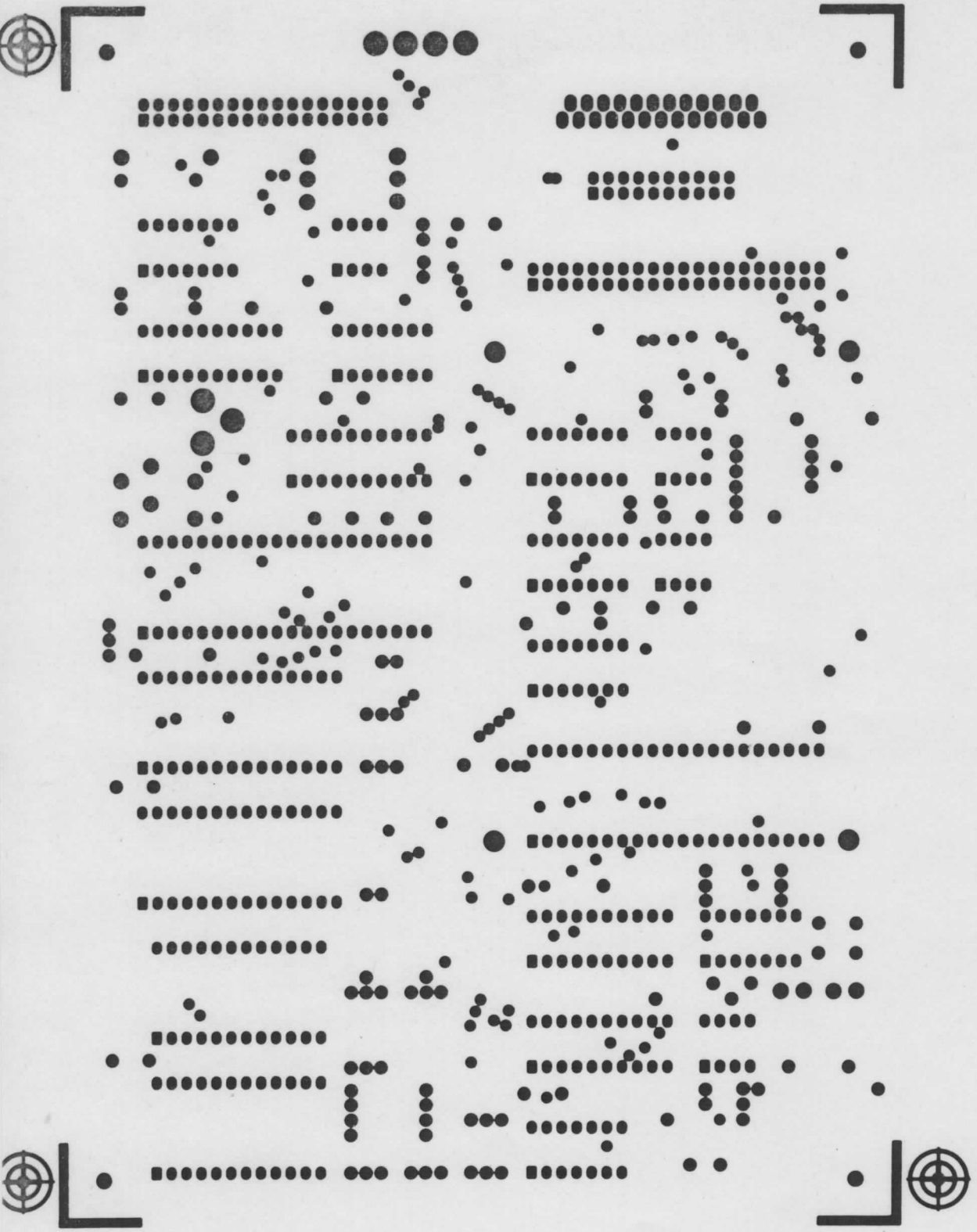
U12 WD2793

For test, install jumper at Pin 22 to Ground.
Enable (1) or Disable (0) precompensation at Pin 1 per disk drive vendor recommendation.

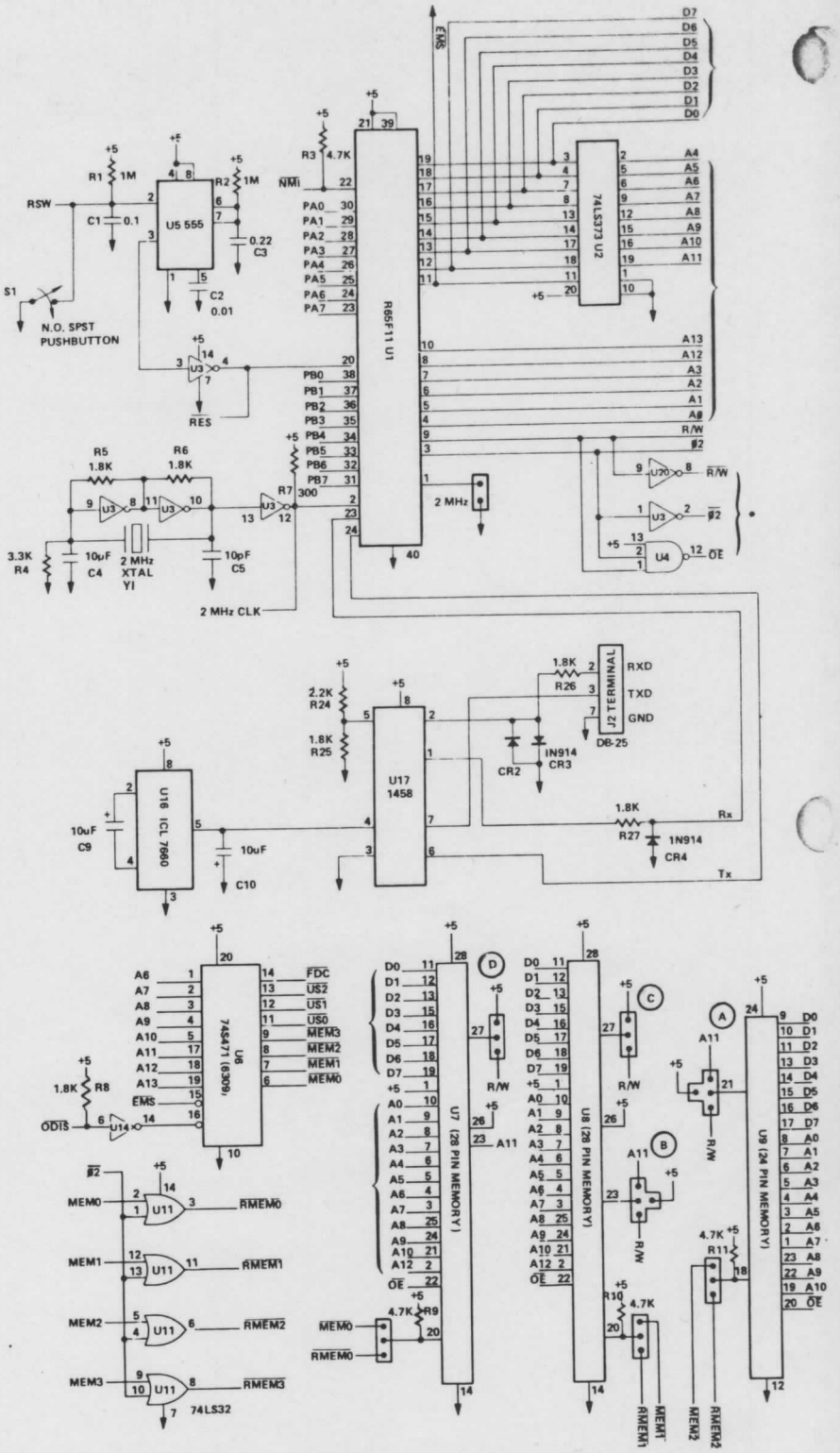
Note:

For location of jumpers see schematic, silkscreen, or board artwork.

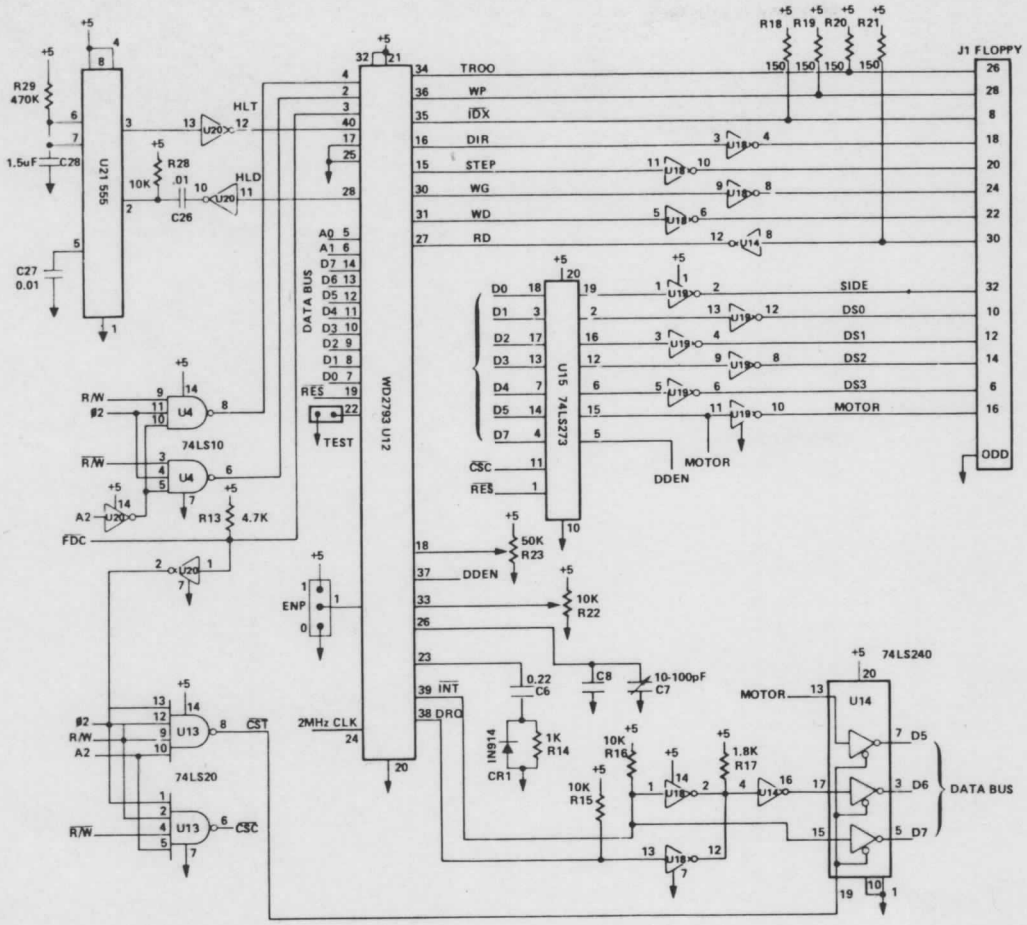

REDUCE TO 7.550 ± .005
SILK SCREEN
TB1
GND +5V
J1
J2
J3 J4
U1
U2
U3
U4
U5
U6
U7
U8
U9
U10
U11
U12
U13
U14
U15
U16
U17
U18
U19
U20
U21
Y1
S1
RSC FORTH
Rockwell
ENP
R3
R7
R13
R15
R16
R17
R18
R19
R20
R21
R22 R23
R24
R25
R26
R27
R28
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
C13
C14
C15
C16
C17
C18
C19
C26
C27
C28
CR1
CR2
CR3
CR4
2M
M0
M1
M2
M3
r0
r1
r2
r3
+5 C r/w
+5 D r/w
+5 E r/w
A11 +5 r/w
A11 +5 r/w

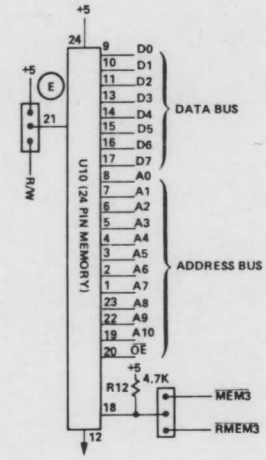


SOLDER MASK



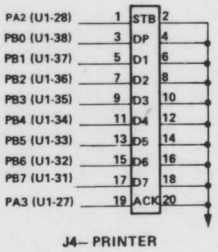
* Set Video Display Terminal for: 7 Data, No Parity, 2 Stop @ 1200 Baud.



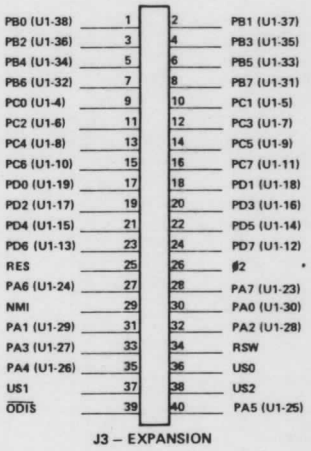


0.1 uF BYPASS CAPACITORS

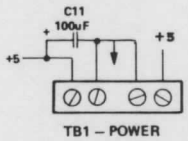
C12
C13
C14
C15
C16
C17
C18



J4– PRINTER

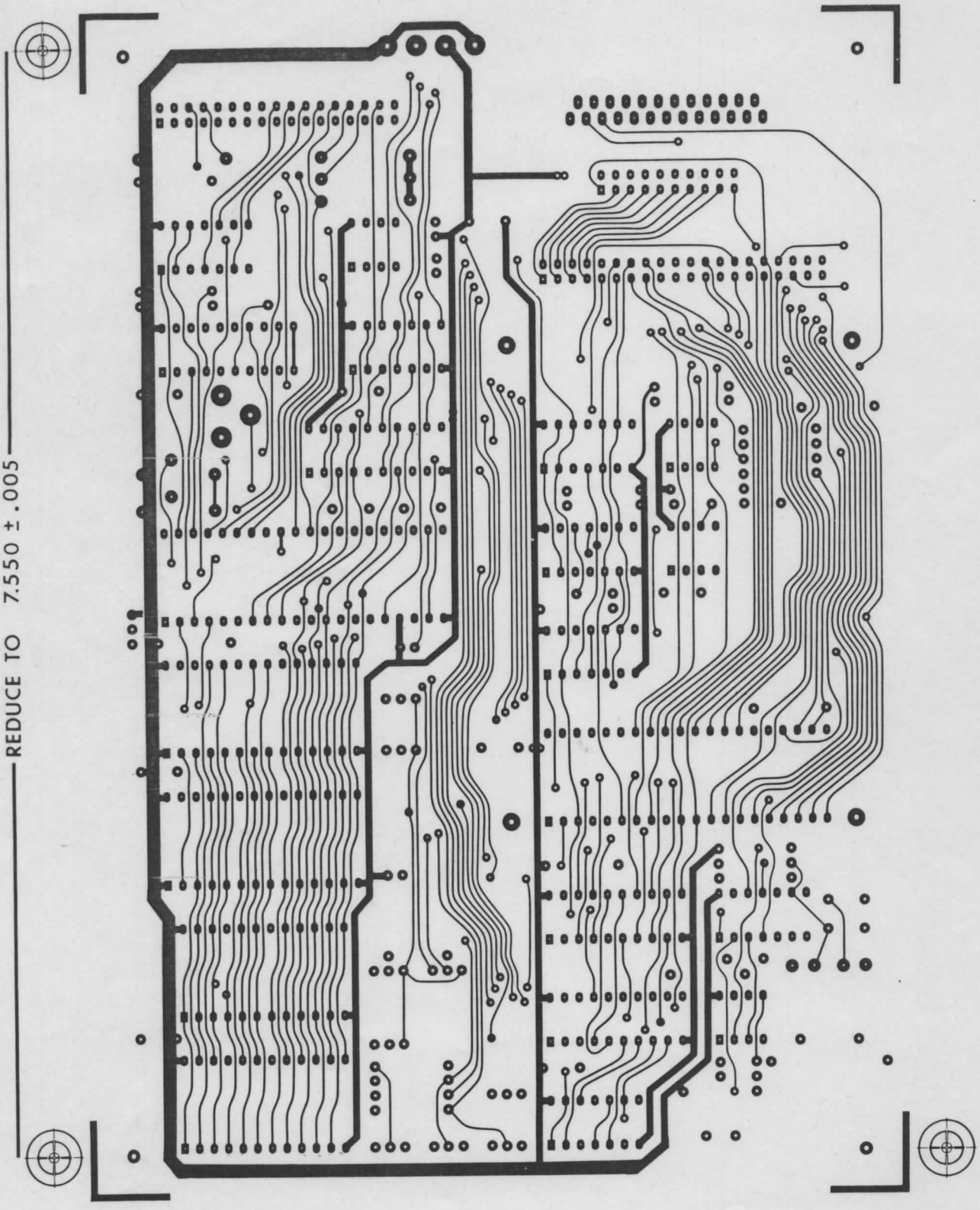


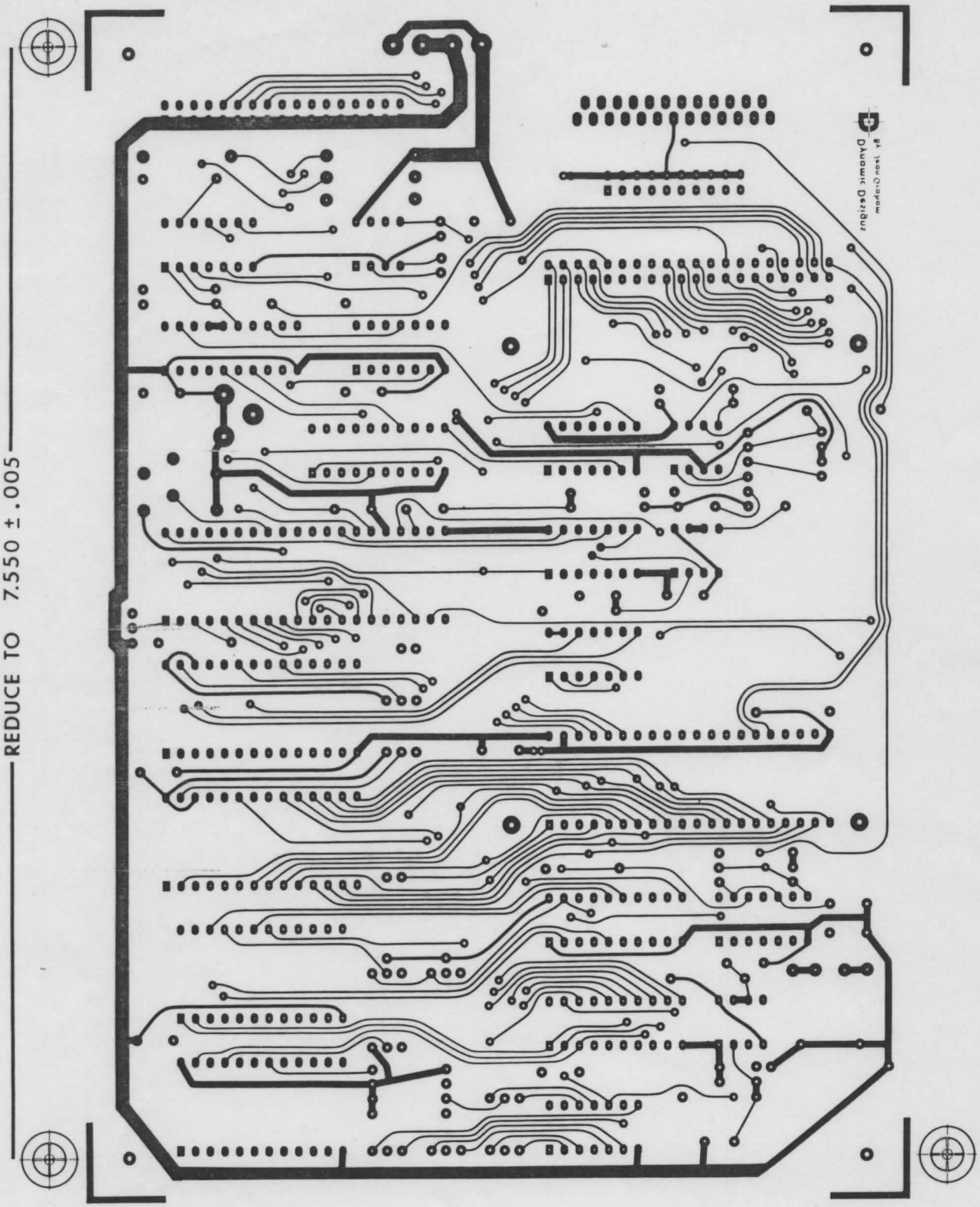
J3 – EXPANSION

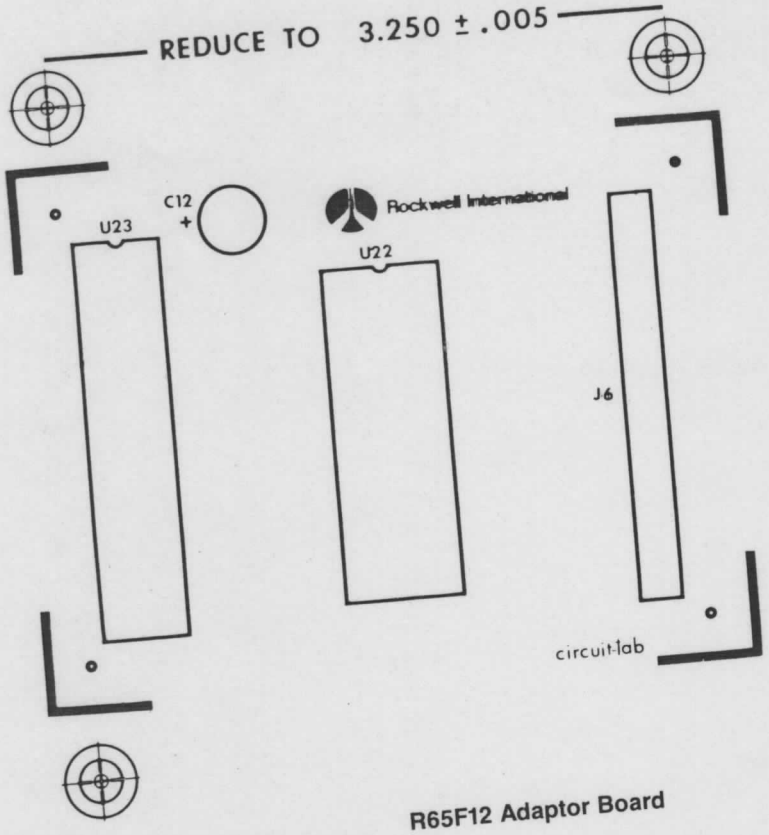


TB1 – POWER

RSC-FORTH Development System

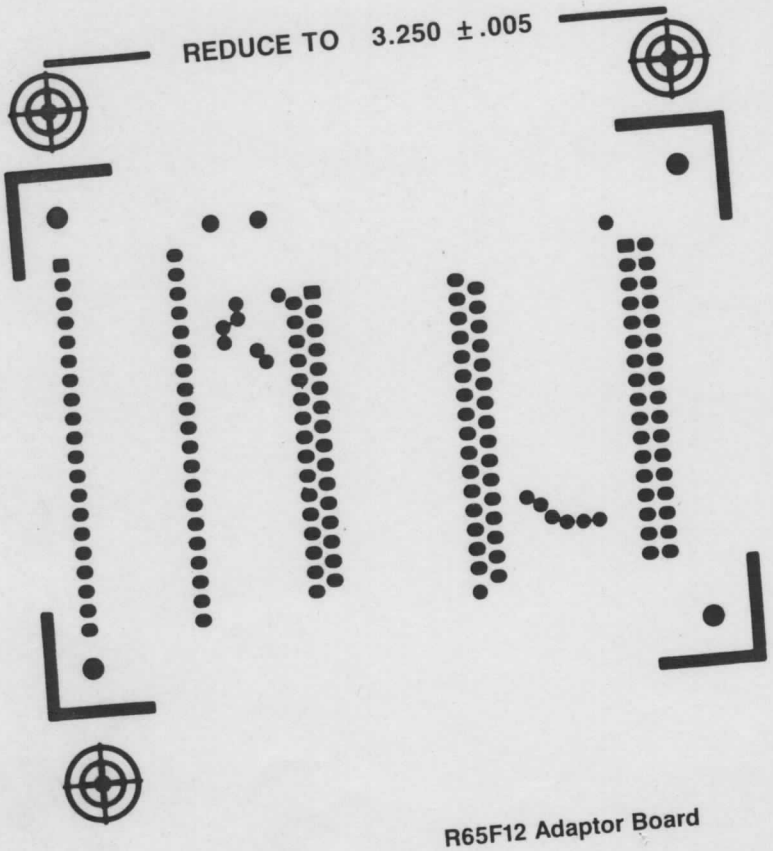

REDUCE TO 7.550 ± .005
COMPONENT SIDE


REDUCE TO 7.550 ± .005
SOLDER SIDE



SILK SCREEN

R65F12 Adaptor Board



SOLDER MASK

R65F12 Adaptor Board

Parts List

Description	Part Number (Vendor)	Description	Part Number (Vendor)
R65F11 Development Board			
U1	R65F11 (Rockwell)	R4	3.3 K Ohm
U2	74LS373 (4)	R5, R6, R8, R17, R25–R27	1.8 K Ohm
U3, U20	74LS04 (4)	R7	300 Ohm
U4	74LS10 (4)	R14	1.0 K Ohm
U5, U21	555	R15, R16, R28	10 K Ohm
U6	74S471 (6309) (4)	R18–R21	150 Ohm (½W)
U7	R65FR1 (Rockwell)	R22	10 K Ohm POT (Bourns 3009P-1-103)
U8	28-Pin Memory	R23	50 K Ohm POT (Bourns 3009-1-503)
U9	24-Pin Memory	R24	2.2 K Ohm
U10	24-Pin Memory	R29	470 K Ohm
U11	74LS32 (4)	CR1-CR4	1N914 or eq
U12	WD2793 (Western Digital)	C1, C12–C19	0.1 μF
U13	74LS20 (4)	C2, C26, C27	0.01 μF
U14	74LS240 (4)	C3, C6	0.22 μF
U15	74LS273 (4)	C4, C5	10 pF
U16	ICL7660 (Intersil)	C7	10–100 pF variable (Sprague GKF70000)
U17	LM1458		
U18, U19	7406	C8	Optional
J1	34 pin (3M #3431-2002)	C9, C10	10 μF electrolytic
J2	DB25 (AMP 2065-84-2)	C11	100 μF electrolytic
J3	40 pin (3M #3432-2002)	C19–25	Not Used
J4	20 pin (3M #3428-2002)	C28	1.5 μF
TB1	0.2″ center terminal strip (Buchanan #SSB404)	S1	N.O. Pushbutton (Chicago Switch EIA50)
R1, R2	1.0 M Ohm	Y1	2.000 MHz Crystal
R3, R9–R13	4.7 K Ohm		
R65F12 Adaptor Board			
U22	R65F12 (Rockwell)	C12	100 μF electrolytic
U23	Wire Wrap Pins (40 ea.)	J6	34 pin (3M #3431-2002)

Notes:

1. All resistors are ¼ W, 10% carbon unless otherwise noted.
2. All capacitors are 16 V ceramic disks unless otherwise noted.
3. All ICs should be socketed: 40, 28, and 24 pin — 2 each; 20 and 8 pin — 4 each; 14 pin — 7 each.
4. 74LS devices should be purchased to Texas Instruments specifications or equivalent.

SEMICONDUCTOR PRODUCTS DIVISION REGIONAL ROCKWELL SALES OFFICES

HOME OFFICE
Semiconductor Products Division
Rockwell International
4311 Jamboree Road
Newport Beach, California 92660

Mailing Address:
P.O. Box C
Newport Beach, California 92660
Mail Code 501-300
Tel: 714-833-4700
TWX: 910 591-1698

UNITED STATES
Semiconductor Products Division
Rockwell International
1842 Reynolds
Irvine, California 92714
(714) 833-4655
ESL 62108710
TWX: 910 595-2518

Semiconductor Products Division
Rockwell International
921 Bowser Road
Richardson, Texas 75080
(214) 996-6500
Telex: 73-307

Semiconductor Products Division
Rockwell International
10700 West Higgins Rd., Suite 102
Rosemont, Illinois 60018
(312) 297-8862
TWX: 910 233-0179 (RI MED ROSM)

Semiconductor Products Division
Rockwell International
5001B Greentree
Executive Campus, Rt. 73
Marlton, New Jersey 08053
(609) 596-0090
TWX: 710 940-1377

FAR EAST
Semiconductor Products Division
Rockwell International Overseas Corp
Itohpia Hirakawa-cho Bldg
7-6, 2-chome, Hirakawa-cho
Chiyoda-ku, Tokyo 102, Japan
(03) 265-8806
Telex: J22198

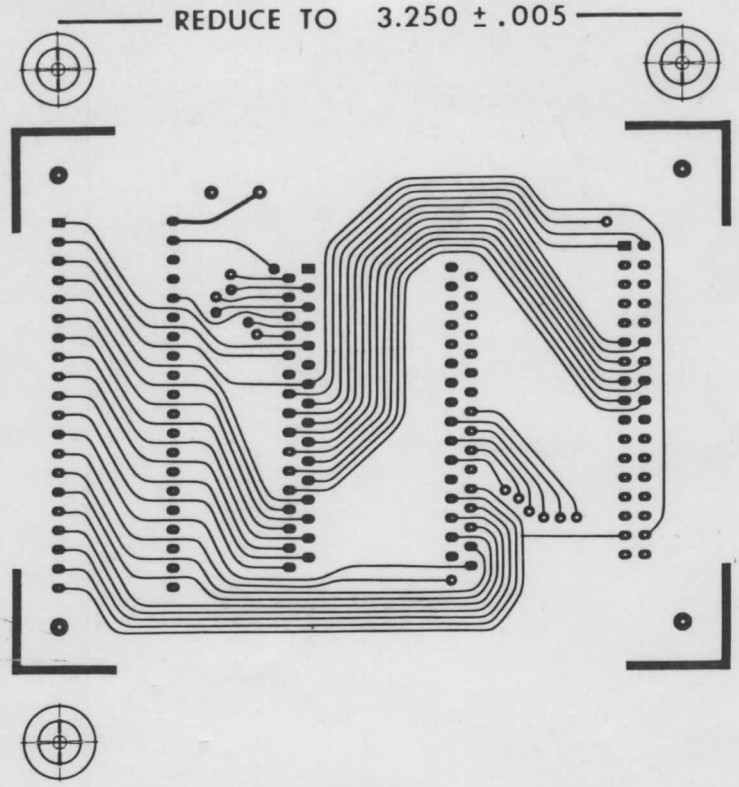
EUROPE
Semiconductor Products Division
Rockwell International GmbH
Fraunhoferstrasse 11
D-8033 Munchen-Martinsried
West Germany
(089) 857-6016
Telex: 0521/2650 rimd d

Semiconductor Products Division
Rockwell International
Heathrow House, Bath Rd.
Cranford, Hounslow,
Middlesex, England
(01) 759-9911
Telex: 851-25463

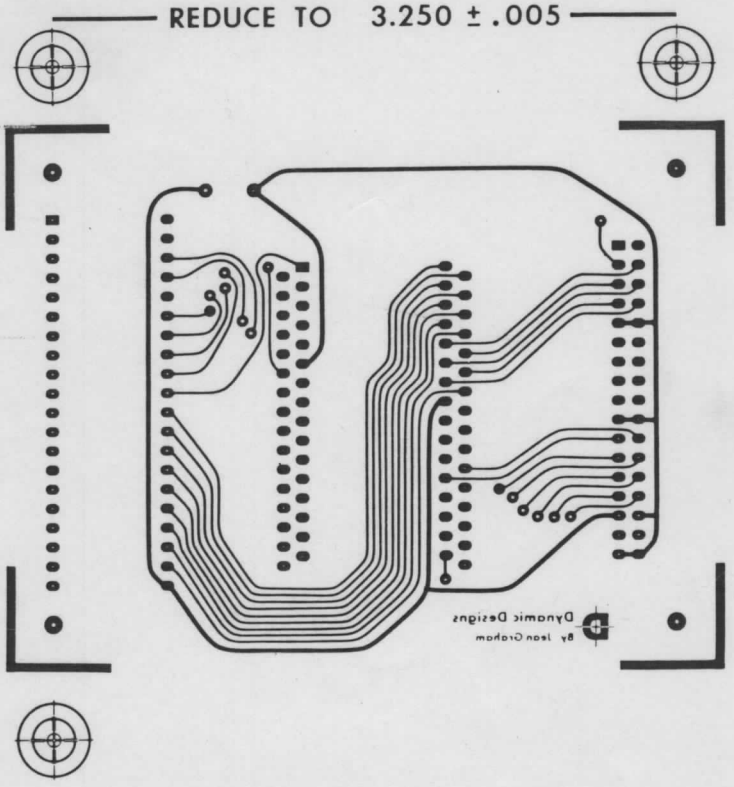
Semiconductor Products Division
Rockwell Collins
Via Boccaccio, 23
20123 Milano, Italy
498.74.79
Telec. 202/82

YOUR LOCAL REPRESENTATIVE

8/83



COMPONENT SIDE



SOLDER SIDE

ROM 95'000
32'000
magazzino
48'000

R65F11 AND R65F12 FORTH BASED MICROCOMPUTERS PRODUCT DESCRIPTION

SECTION 1 INTRODUCTION

1.1 FEATURES

- FORTH kernel in ROM
- Enhanced 6502 CPU
  - —Four new bit manipulation instructions:
    - Set Memory Bit (SMB)
    - Reset Memory Bit (RMB)
    - Branch on Bit Set (BBS)
    - Branch on Bit Reset (BBR)
  - —Decimal and binary arithmetic modes
  - —13 addressing modes
  - —True indexing
- 192-byte static RAM
- 16 bidirectional, TTL-compatible I/O lines (two ports, R65F11) or 40 bidirectional, TTL-compatible I/O lines (five ports, R65F12)
- One 8-bit port with programmable latched input
- Two 16-bit programmable counter/timers, with latches
  - —Pulse width measurement
  - —Asymmetrical pulse generation
  - —Pulse generation
  - —Interval timer
  - —Event counter
  - —Retriggerable interval timer
- Serial port
  - —Full-duplex asynchronous operation mode
  - —Selectable 5- to 8-bit characters
  - —Wake-up feature
  - —Synchronous shift register mode
  - —Standard programmable bit rates, programmable up to 62.5K bits/sec
- Ten interrupts
  - —Four edge-sensitive lines; two positive, two negative
  - —Reset
  - —Non-maskable
  - —Two counter
  - —Serial data received
  - —Serial data transmitted
- Expandable to 16K bytes of external memory
- Flexible clock circuitry
  - —2-MHz or 1-MHz internal operation
  - —Internal clock with external XTAL at two times internal frequency
  - —External clock input divided by one or two
- 1 μs minimum instruction execution time @ 2 MHz (2vidi)
- NMOS silicon gate, depletion load technology
- Single +5V power supply
- 12 mW standby power for 32 bytes of the 192-byte RAM
- 40-pin DIP (R65F11)
- 64-pin QUIP (R65F12) has three additional 8-bit I/O ports to provide a total of 40 I/O lines.

1.2 SUMMARY

The Rockwell R65F11 and R65F12 are complete, high-performance 8-bit NMOS single chip microcomputers, and are compatible with all members of the R6500 family.

The kernel of the high level Rockwell Single Chip RSC-FORTH language is contained in the preprogrammed ROM of the R65F11 and R65F12. RSC-FORTH is based on the popular fig-FORTH model with extensions. All of the run time functions of RSC-FORTH are contained in the ROM, including 16- and 32-bit mathematical, logical and stack manipulation, plus memory and input/output operators. The RSC-FORTH Operating System allows an external user program written in RSC-FORTH or Assembly Language to be executed from external EPROM, or development of such a program under the control of the R65FR1 RSC-FORTH Development ROM. Other Development ROM's can also be accommodated.

The R65F11 and R65F12 consist of an enhanced 6502 CPU, an internal clock oscillator, 192 bytes of Random Access Memory (RAM) and versatile interface circuitry. The interface circuitry includes two 16-bit programmable timer/counters, 16 bidirectional input/output lines (including four edge-sensitive lines and input latching on one 8-bit port), a full-duplex serial I/O channel, ten interrupts and bus expandability.

The innovative architecture and the demonstrated high performance of the R6502 CPU, as well as instruction simplicity, results in system cost-effectiveness and a wide range of com-

Document No. 29651N49
Order No. 2146
February 1983

computational power. These features in combination with the FORTH high level operating system make the R65F11 and R65F12 ideal for microcomputer applications.

For systems requiring additional I/O ports, the 64-pin QUIP version, the R65F12, provides three additional 8-bit ports.

A complete RSC-FORTH development system can be created with three MOS parts: the R65F11, one RAM chip and the R65FR1 Development ROM.

This product description is for the reader familiar with the R6502 CPU hardware and programming capabilities. A detailed description of the R6502 CPU hardware is included in the R6500 Microcomputer System Hardware Manual (Document Number 29650N31). A description of the instruction capabilities of the R6502 CPU is contained in the R6500 Microcomputer System Programming Manual (Document Number 29650N30).

1.3 ORDERING INFORMATION

Part No.	Description
R65F11	40-Pin FORTH Based Microcomputer
R65F12	64-Pin FORTH Based Microcomputer
R65FR1	FORTH Development ROM for R65F11 or R65F12
Order No.	**Description**
2148	FORTH Based Microcomputer User's Manual*
NOTE: *Included with R65FR1.	

SECTION 2
INTERFACE REQUIREMENTS

This section describes the interface requirements for the R65F11 and R65F12 single chip microcomputers. Figure 2-1 is the Interface Diagram for the R65F11 and R65F12, Figure 2-2 shows the pin out configuration and Table 2-1 describes the function of each pin of the R65F11 and R65F12. Figure 3-1 is a detailed block diagram.

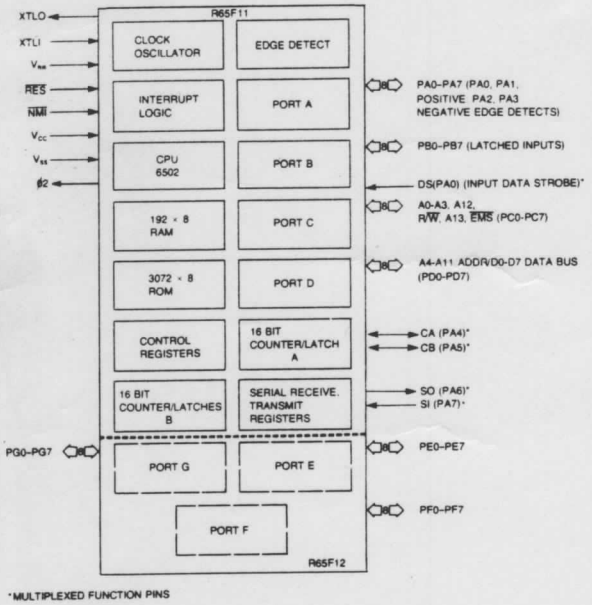
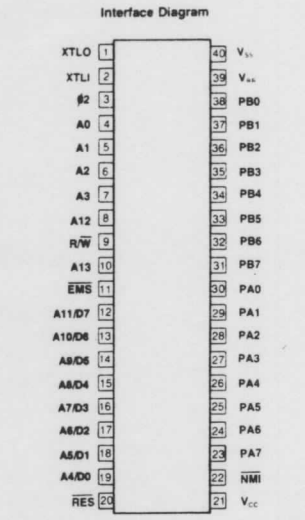


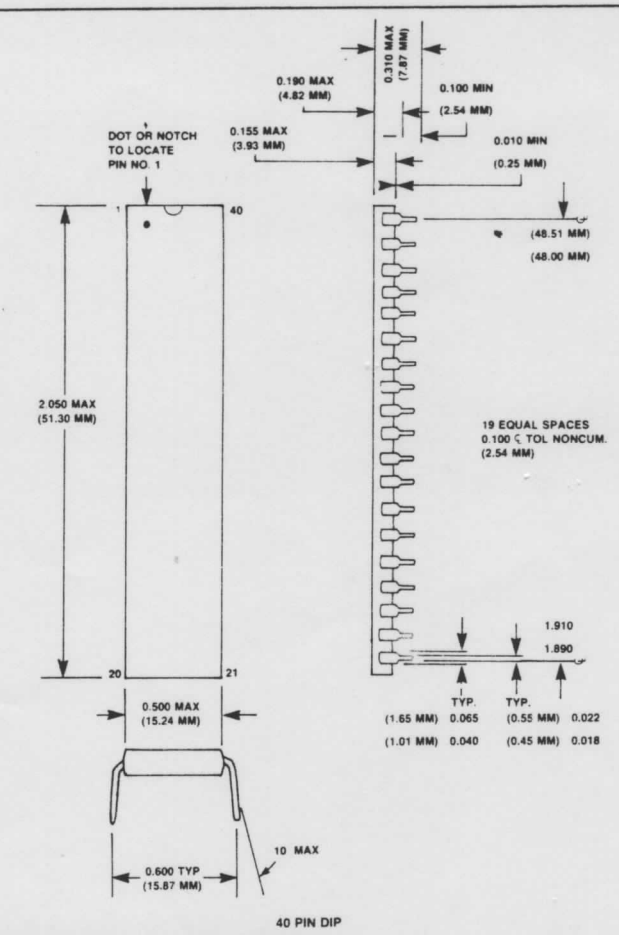
FIGURE 2-1. R65F11 and R65F12 Interface Diagram

TABLE 2-1. R65F11 and R65F12 Pin Descriptions

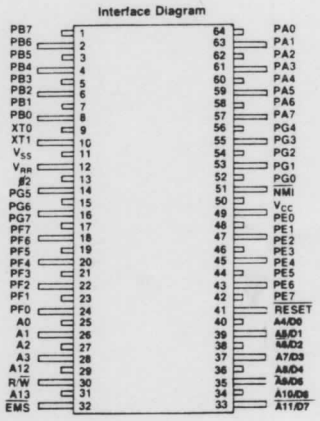
SIGNAL NAME	PIN NO. R65F11	PIN NO. R65F12	DESCRIPTION
V_{CC}	21	50	Main power supply +5V
V_{RR}	39	12	Separate power pin for RAM. In the event that V_{CC} power is lost, this power retains RAM data.
V_{SS}	40	11	Signal and power ground (0V)
XTLI	2	10	Crystal or clock input for internal clock oscillator. Also allows input of X1 clock signal if XTLO is connected to V_{SS}, or X2 or X4 clock if XTLO is floated.
XTLO	1	9	Crystal output from internal clock oscillator.
$\overline{RES}$	20	41	The Reset input is used to initialize the R65F11. This signal must not transition from low to high for at least eight cycles after V_{CC} reaches operating range and the internal oscillator has stabilized.
$\phi 2$	3	13	Clock signal output at internal frequency.
$\overline{NMI}$	22	51	A negative going edge on the Non-Maskable Interrupt signal requests that a non-maskable interrupt be generated within the CPU.
PA0-PA7 PB0-PB7	30-23 38-31	57-64 1-8	Two 8-bit ports used for either input/output. Each line of Ports A and B consist of an active transistor to V_{SS} and a passive pull-up to V_{CC}.
PC0-PC7 A0-A3 A12, R/$\overline{W}$ A13, $\overline{EMS}$	4-11	25-32	Port C has an active pull-up transistor. Port D has active pull-up and pull-down transistors. Ports C and D lines form the external multiplexed address and data bus to allow external memory addressing.
PD0-PD7 A4-A11 D0-D7	19-12	33-40	
PE0-PE7 PF0-PF7 PG0-PG7		42-49 24-17 14-16, 52-56	On the R65F12, Port E may be used for output only. Ports F and G are similar to Ports A and B in construction and may be used for inputs or outputs.



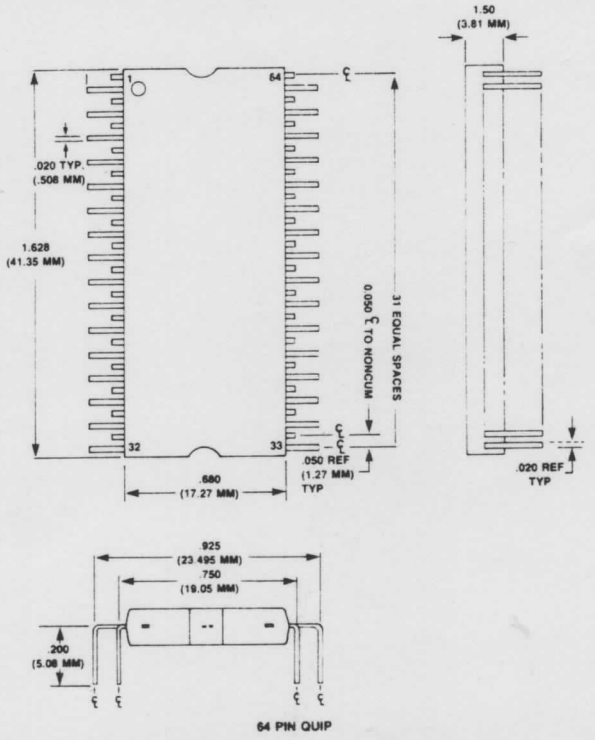
R65F11 Pin Out Designation



R65F11 Dimensional Outline



R65F12 Pin Out Designation



R65F12 Dimensional Outline

FIGURE 2-2. Pin Out Configuration

SECTION 3
SYSTEM ARCHITECTURE

This section provides a functional description of the R65F11 and R65F12. Functionally the R65F11 consists of a CPU, RAM memory, two 8-bit parallel I/O ports (five in the 64-pin R65F12), a serial I/O port, dual counter/latch circuits, a mode control register, an interrupt flag/enable dual register circuit, and an internal Operating System. The kernel of FORTH in ROM complements the system hardware. A block diagram of the system is shown in Figure 3-1.

NOTE

Throughout this document, unless specified otherwise, all memory or register address locations are specified in hexadecimal notation.

3.1 CPU LOGIC

The R65F11 internal CPU is a standard 6502 configuration with an 8-bit Accumulator register, two 8-bit Index Registers (X and Y); an 8-bit Stack Pointer register, and ALU, a 16-bit Program Counter, and standard instruction register/decode and internal timing control logic.

3.1.1 Accumulator

The accumulator is a general purpose 8-bit register that stores the results of most arithmetic and logic operations. In addition, the accumulator usually contains one of the two data words used in these operations.

3.1.2 Index Registers

There are two 8-bit index registers, X and Y. Each index register can be used as a base to modify the address data program counter and thus obtain a new address—the sum of the program counter contents and the index register contents.

When executing an instruction which specifies indirect addressing, the CPU fetches the op code and the address, and modifies the address from memory by adding the index register to it prior to loading or storing the value of memory.

Indexing greatly simplifies many types of programs, especially those using data tables.

3.1.3 Stack Pointer

The Stack Pointer is an 8-bit register. It is automatically incremented and decremented under control of the microprocessor to perform stack manipulation in response to either user instructions, an internal $\overline{IRQ}$ interrupt, or the external interrupt line $\overline{NMI}$. The Stack Pointer must be initialized by the user program.

The stack allows simple implementation of multiple level interrupts, subroutine nesting and simplification of many types of data manipulation. The JSR, BRK, RTI and RTS instructions use the stack and Stack Pointer.

The stack can be envisioned as a deck of cards which may only be accessed from the top. The address of a memory location is stored (or "pushed") onto the stack. Each time data are to be pushed onto the stack, the Stack Pointer is placed on the Address Bus, data are written into the memory location addressed by the Stack Pointer, and the Stack Pointer is decremented by 1. Each time data are read (or "pulled") from the stack, the Stack Pointer is incremented by 1. The Stack Pointer is then placed on the Address Bus, and data are read from the memory location addressed by the Pointer.

The stack is located on zero page, i.e., memory locations 00FF-0040. After reset, which leaves the Stack Pointer indeterminate, normal usage calls for its initialization at 00FF.

3.1.4 Processor Status Register

The 8-bit Processor Status Register contains seven status flags. Some of these flags are controlled by the user program; others may be controlled both by the user's program and the CPU. The R6500 instruction set contains a number of conditional branch instructions which are designed to allow testing of these flags. See Appendix B for details.

3.1.5 Program Counter

The 16-bit Program Counter provides the addresses that are used to step the processor through sequential instructions in a program. Each time the processor fetches an instruction from program memory, the lower (least significant) byte of the Program Counter (PCL) is placed on the low-order bits of the Address Bus and the higher (most significant) byte of the Program Counter (PCH) is placed on the high-order 8 bits of the Address Bus. The Counter is incremented each time an instruction or data is fetched from program memory.

3.1.6 Arithmetic And Logic Unit (ALU)

Each bit of the ALU has two inputs. These inputs can be tied to various internal buses or to a logic zero; the ALU then generates the function (AND, OR, SUM, and so on) using the data on the two inputs.

3.1.7 Instruction Register and Instruction Decode

Instructions are fetched from ROM or RAM and gated onto the Internal Data Bus. These instructions are latched into the Instruction Register then decoded along with timing and interrupt signals to generate control signals for the various registers.

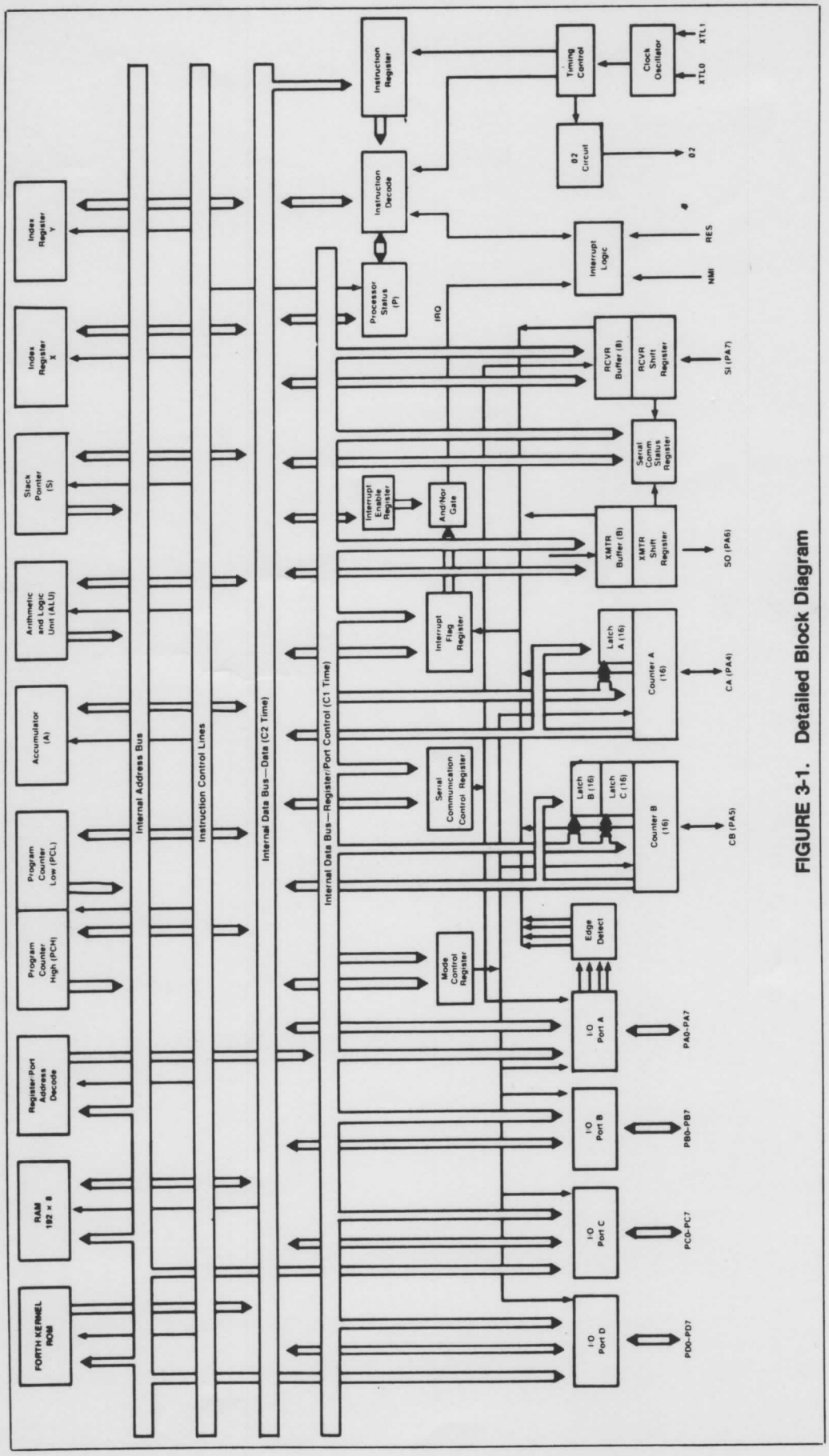


FIGURE 3-1. Detailed Block Diagram

3.1.8 Timing Control

The Timing Control Logic keeps track of the specific instruction cycle being executed. This logic is set to T0 each time an instruction fetch is executed and is advanced at the beginning of each Phase One clock pulse for as many cycles as are required to complete the instruction. Each data transfer which takes place between the registers is caused by decoding the contents of both the instruction register and timing control unit.

3.1.9 Interrupt Logic

Interrupt logic controls the sequencing of three interrupts; $\overline{RES}$, $\overline{NMI}$ and $\overline{IRQ}$. $\overline{IRQ}$ is generated by any one of eight conditions: 2 Counter Overflows, 2 Positive Edge Detects, 2 Negative Edge Detects, and 2 Serial Port Conditions.

3.2 CPU INSTRUCTION SET

The machine code instruction set of the R65F11 and R65F12 microcomputers are based on the popular R6500 microprocessor set. They contain all the instructions in the standard R6502 set, with the addition of the four new bit instructions added to the R6511 processor family. Refer to Appendix A for the Op Code mnemonics addressing matrix for details on these instructions.

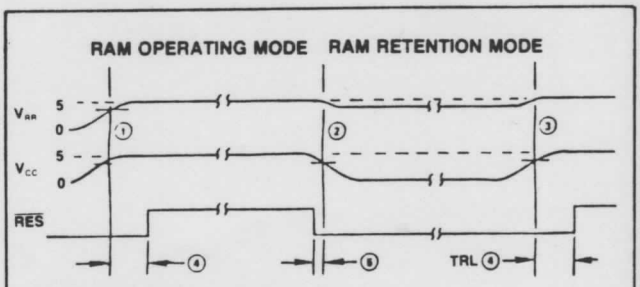
3.3 READ-ONLY-MEMORY (ROM)

The ROM consists of preprogrammed memory with an address space from F400 to FFFF. It contains the run time kernel of the high level language Rockwell Single Chip FORTH. There are 133 included functions stored in the ROM. Codes are in the format of a two byte code field, which identifies the interpreter assigned to execute that word, followed by a variable length Parameter Field, which contains the instructions and data used by that interpreter according to the programmed intention of that definition. See Appendix D for a complete list of the names of all included words. All words needed for support of the run time operation of dedicated applications programs are included. The RSC-FORTH Operating System is also part of the ROM code and is entered upon Reset. This Operating System allow the R65F11 and R65F12 to auto start a user program written in either RSC-FORTH or Assembly Language or enter a Development ROM if one is present. If no auto start program is found, an attempt will be made to boot an operating program from floppy disk.

3.4 RANDOM ACCESS MEMORY (RAM)

The RAM consists of 192 bytes of read/write memory with an assigned page zero address of 0040 through 00FF. The R65F11 and R65F12 provide a separate power pin (V_{RR}) which may be used for standby power for 32 bytes located at 0040-005F. In the event of the loss of V_{CC} power, the lowest 32 bytes of RAM data will be retained if standby power is supplied to the V_{RR} pin. If the RAM data retention is not required then V_{RR} must be connected to V_{CC}. During operation V_{RR} must be at the V_{CC} level.

For the RAM to retain data upon loss of V_{CC}, V_{RR} must be supplied within operating range and $\overline{RES}$ must be driven low at least eight Ø2 clock pulses before V_{CC} falls out of operating range. $\overline{RES}$ must then be held low while V_{CC} is out of operating range and until at least eight Ø2 clock cycles after V_{CC} is again within operating range and the internal Ø2 oscillator is stabilized. V_{RR} must remain within V_{CC} operating range during normal operation. When V_{CC} is out of operating range, V_{RR} must remain within the V_{RR} retention range in order to retain data. Figure 3-2 shows typical waveforms.



1 INITIAL APPLICATION OF V_{CC} AND V_{RR}.
2 LOSS OF V_{CC}, RAM ON STANDBY POWER.
3 REAPPLICATION OF V_{CC}.
4 >8 Ø2 CLOCK PULSES AFTER OSCILLATOR STABILIZATION.
5 >8 Ø2 CLOCK PULSES.

FIGURE 3-2. Data Retention Timing

3.5 CLOCK OSCILLATOR

A reference frequency can be generated with the on-chip oscillator using an external crystal. The oscillator reference frequency passes through an internal countdown network (divide by 2) to obtain the internal operating frequency (see Figure 3-3a).

Internal timing can also be controlled by driving the XTLI pin with an external frequency source. Figure 3-3b shows typical connections. If XTLO is left floating, the external source is divided by the internal countdown network. However, if XTLO is tied to V_{SS}, the internal coundown network is bypassed causing the chip to operate at the frequency of the external source.

The R65F11 and R65F12 operate in the CLOCK MASTER mode. In this mode a frequence source (crystal or external source) must be applied to the XTLI and XTLO pins. Ø2 is a buffered output signal which closely approximates the internal timing. When a common external source is used to drive muliple devices the internal timing between devices as well as their Ø2 outputs will be skewed in time. If skewing represents a system problem it can be avoided by the Master/Slave connection and options shown in Figure 3-4.

The R65F11 and R65F12 is operated in the CLOCK MASTER MODE. A second processor could be operated in the CLOCK SLAVE MODE. Mask options in the SLAVE unit convert the Ø2 signal into a clock input pin which is tightly coupled to the internal timing generator. As a result the internal timing of the MASTER and SLAVE units are synchronized with minimum

skew. If the ø2 signal to the SLAVE unit is inverted, the MASTER and SLAVE UNITS WILL OPERATE OUT. OF PHASE. This approach allows the two devices to share external memory using cycle stealing techniques.

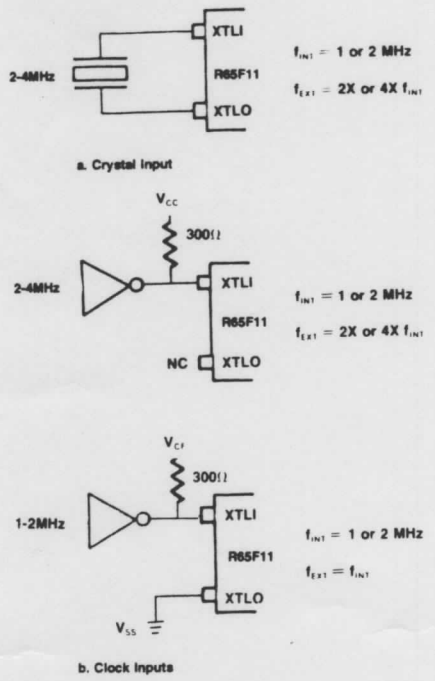


FIGURE 3-3. Clock Oscillator Input Options

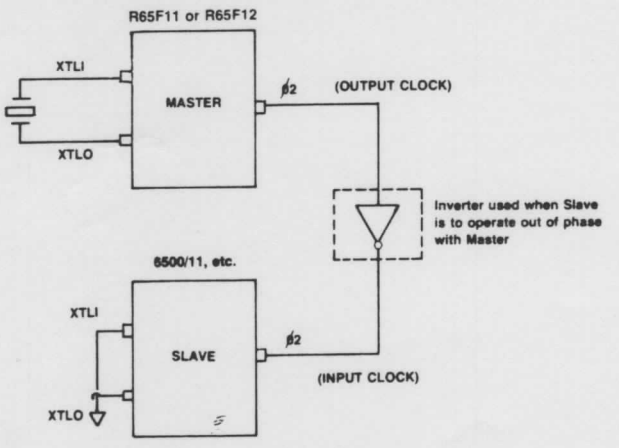


FIGURE 3-4. Master/Slave Connections

3.6 MODE CONTROL REGISTER (MCR)

The Mode Control Register contains control bits for the multifunction I/O ports and mode select bits for Counter A and Counter B. Its setting, along with the setting of the Serial Communications Control Register (SCCR), determines the basic configuration of the R65F11 and R65F12 in any application. The Mode Control Register bit assignment is shown in Figure 3-5. MCR Bits 7, 6, 5 must remain 1's in order for external memory referencing to be enabled.

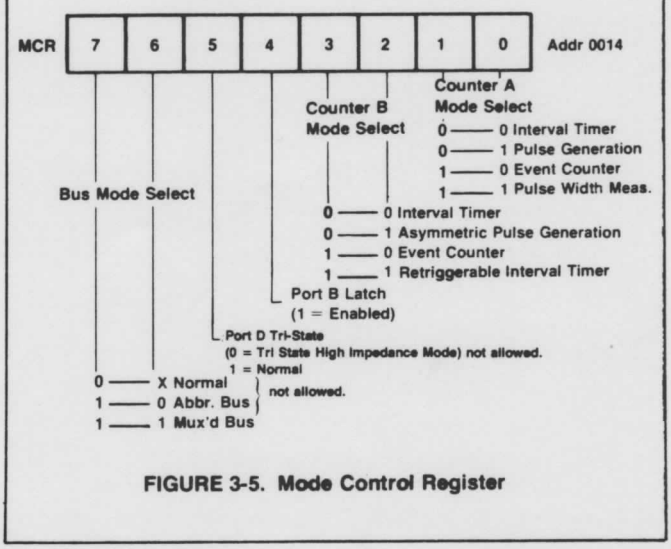


FIGURE 3-5. Mode Control Register

The use of Counter A Mode Select is shown in Section 6.1.

The use of Counter B Mode Select is shown in Section 6.2.

The use of Port B Latch Enable is shown in Section 4.4.

3.7 INTERRUPT FLAG REGISTER (IFR) AND INTERRUPT ENABLE REGISTER (IER)

An $\overline{IRQ}$ interrupt request can be initiated by any or all of eight possible sources. These sources are all capable of being enabled or disabled by the use of the appropriate interrupt enabled bits in the Interrupt Enable Register (IER). Multiple simultaneous interrupts will cause the $\overline{IRQ}$ interrupt request to remain active until all interrupting conditions have been serviced and cleared.

The Interrupt Flag Register contains the information that indicates which I/O or counter needs attention. The contents of the Interrupt Flag Register may be examined at any time by reading at address: 0011. Edge detect IFR bits may be cleared in low level code by executing a RMB instruction at address location 0010. The RMB X, (0010) instruction reads FF, modifies bit X to a "0", and writes the modified value at address location 0011. In this way IFR bits set to a "1" after the read cycle of a Read-Modify-Write instruction (such as RMB) are protected from being cleared. A logic "1" is ignored when writing to edge detect IFR bits.

Each IFR bit has a corresponding bit in the Interrupt Enable Register which can be set to a "1" by writing a "1" in the respective bit position at location 0012. Individual IER bits may be cleared by writing a "0" in the respective bit position, or by $\overline{RES}$. If set to a "1", an IRQ will be generated when the corresponding IFR bit becomes true. The Interrupt Flag Register and Interrupt Enable Register bit assignments are shown in Figure 3-6 and the functions of each bit are explained in Table 3-1.

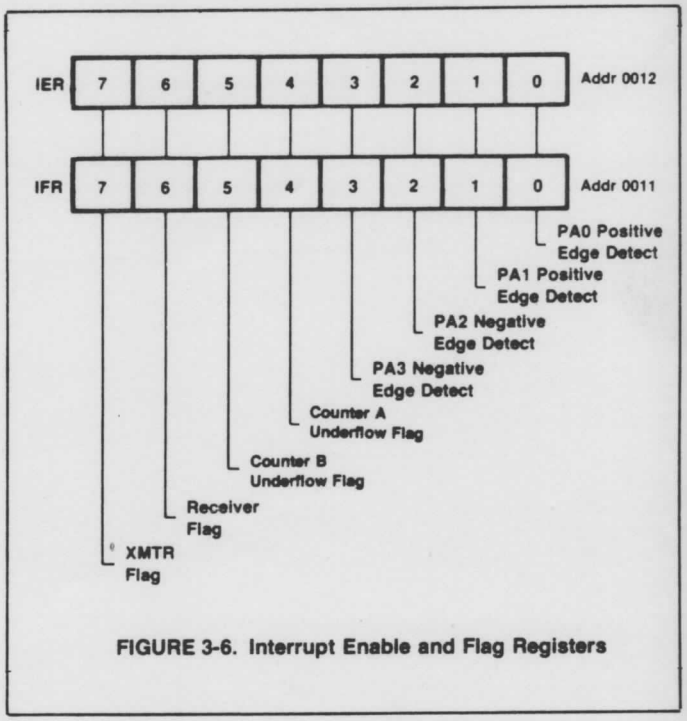


FIGURE 3-6. Interrupt Enable and Flag Registers

TABLE 3-1. Interrupt Flag Register Bit Codes

BIT CODE	FUNCTION
IFR 0:	PA0 Positive Edge Detect Flag—Set to a "1" when a positive going edge is detected on PA0. Cleared by RMB O (0010) instruction or by $\overline{RES}$.
IFR 1:	PA1 Positive Edge Detect Flag—Set to a 1 when a positive going edge is detected on PA1. Cleared by RMB 1 (0010) instruction or by $\overline{RES}$.
IFR 2:	PA2 Negative Edge Detect Flag—Set to a 1 when a negative going edge is detected on PA2. Cleared by RMB 2 (0010) instruction or by $\overline{RES}$.
IFR 3:	PA3 Negative Edge Detect Flag—Set to 1 when a negative going edge is detected on PA3. Cleared by RMB 3 (0010) instruction or by $\overline{RES}$.
IFR 4:	Counter A Underflow Flag—Set to a 1 when Counter A underflow occurs. Cleared by reading the Lower Counter A at location 0018, by writing to address location 001A, or by $\overline{RES}$.
IFR 5:	Counter B Underflow Flag—Set to a 1 when Counter B underflow occurs. Cleared by reading the Lower Counter B at location 001C, by writing to address location 001E, or by $\overline{RES}$.
IFR 6:	Receiver Interrupt Flag—Set to a 1 when any of the Serial Communication Status Register bits 0 through 3 is set to a 1. Cleared when the Receiver Status bits (SCSR 0-3) are cleared or by $\overline{RES}$.
IFR 7:	Transmitter Interrupt Flag—Set to a 1 when SCSR 6 is set to a 1 while SCSR 5 is a 0 or SCSR 7 is set to a 1. Cleared when the Transmitter Status bits (SCSR 6 & 7) are cleared or by $\overline{RES}$.

3.8 OPERATING SYSTEM

The system startup function, COLD, is executed upon Reset. COLD, a high level FORTH word, forms the basis of the RSC Operating System. Upon reset this function initializes the R65F11 or R65F12 registers to establish the external 16K byte memory map and disable all interrupt sources. It also sets up the serial channel for 1200 baud (assuming a 1 MHz internal clock) asynchronous transmission (seven bits, parity disabled). The internal FORTH structure "W" is prepared for use and the low level input/output vectors are forced to point to the system serial channel routines. The FORTH User Area Pointer, UP, is assigned the value 0300 Hex.

A test is made of the variable CLD/WRM in memory location 030E. If this contains a value other than A55A Hex a cold reset is assumed. In this case, the low level IRQ vector, IRQVEC; the low level NMI Vector, NMIVEC, and the high level interrupt vector, INTVEC, are all forced to point to the system reset routine. This prevents an unintentionally generated interrupt from crashing the system. System variables TIB, RO, SO, UC/L, UPAD, UR/W and BASE are also initialized to their default values.

Whether a warm or cold reset, the memory map is then searched at every 1K byte boundary starting at location 0400 Hex. The first two bytes at each boundary are checked against an A55A Hex bit pattern. This pattern indicates that an auto start program is installed. The next two bytes are assumed to point to the Parameter Field of the high level RSC-FORTH word to be executed upon reset. This may be the main function of a user defined program or the start up routine of a Development ROM. Figure 3-7 details proper alignment.

If no auto start ROM is found, the Operating System turns control over to a program that issues a "NO ROM" message to the systems terminal via the serial channel and attempts to boot a program from disk. A floppy disk controller, compatible with the WD1793 type, is assumed to be present at address 0100 Hex. The first half of Track 0 Sector 1 is loaded from a double density boot diskette into RAM starting at address 005F. When successfully loaded execution will be turned over to this boot program.

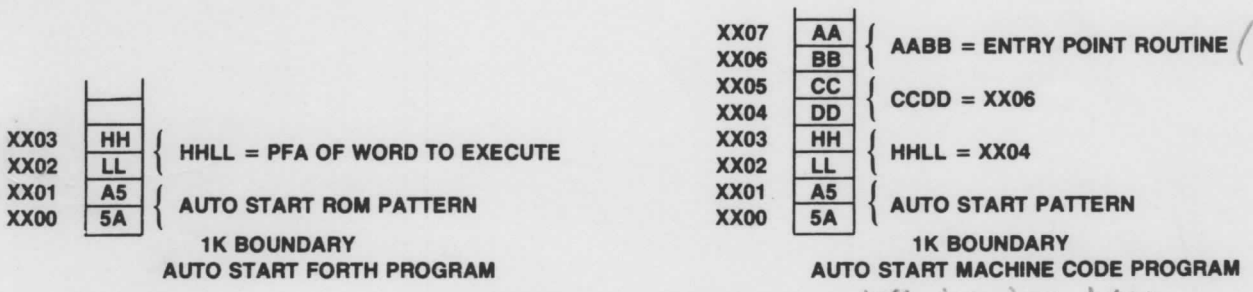


FIGURE 3-7. Auto Start ROM

SECTION 4
PARALLEL INPUT/OUTPUT PORTS

The R65F11 has 16 I/O lines grouped into two 8-bit ports (PA, PB) and 16 lines programmed as an Address/Data bus (PC & PD). Ports A and B may be used either for input or output individually or in groups of any combination. The R65F12 has 24 additional port lines grouped into three 8-bit ports (PE, PF, PG).

Multifunction I/O's such as Port A are protected from normal port I/O instructions when they are programmed to perform a multiplexed function.

Internal pull-up resistors (FET's with an impedance range of 3K ≤ Rpu ≤ 12K ohm) are provided on all port pins.

The direction of the I/O lines are controlled by 8-bit port registers located in page zero. This arrangement provides quick programming access using simple two-byte zero page address instructions. There are no direction registers associated with the I/O ports, which simplifies I/O handling. The I/O addresses are shown in Table 4-1.

TABLE 4-1. I/O PORT ADDRESSES

PORT	ADDRESS
A	0000
B	0001
E	0004
F	0005
G	0006

Appendix F.4 shows the I/O Port Timing.

4.1 INPUTS

Inputs for Ports A and B are enabled by loading logic 1 into all I/O port register bit positions that are to correspond to I/O input lines. A low (<0.8V) input signal will cause a logic 0 to be read when a read instruction is issued to the port register. A high (>2.0V) input will cause a logic 1 to be read. An $\overline{RES}$ signal forces all I/O port registers to logic 1 thus initially treating all I/O lines as inputs.

The status of the input lines can be interrogated at any time by reading the I/O port addresses. Note that this will return the actual status of the input lines, not the data written into the I/O port registers.

Read/Modify/Write instructions can be used to modify the operation of PA and PB. During the Read cycle of a Read/Modify/Write instruction the Port I/O register is read. For all other read instructions the port input lines are read. Read/Modify/Write instructions are: ASL, DEC, INC, LSR, RMB, ROL, ROR, and SMB.

4.2 OUTPUTS

Outputs for Ports A and B are controlled by writing the desired I/O line output states into the corresponding I/O port register bit positions. A logic 1 will force a high (>2.4V) output while a logic 0 will force a low (<0.4V) output.

4.3 PORT A (PA)

Port A can be programmed via the Mode Control Register (MCR) and the Serial Communications Control Register (SCCR) as a standard parallel 8-bit, bit independent, I/O port or as serial channel I/O lines, counter I/O lines, or an input data strobe for the Port B input latch option. Table 4-3 tabulates the control and usage of Port A.

In addition to their normal I/O functions, PA0 and PA1 can detect positive going edges, and PA2 and PA3 can detect negative going edges. A proper transition on these pins will set a corresponding status bit in the IFR and generate an interrupt request if the respective Interrupt Enable Bit is set. The maximum rate at which an edge can be detected is one-half the Ø2 clock rate. Edge detection timing is shown in Appendix F.4.

4.4 PORT B (PB)

Port B can be programmed as an 8 bit, bit independent I/O port. It has a latched input capability which may be enabled or disabled via the Mode Control Register (MCR). Table 4-2 tabulates the control and usage of Port B. An Input Data Strobe signal must be provided thru PA0 when Port B is programmed to be used with latched input option. Input data latch timing for Port B is shown in Appendix F.4.

TABLE 4-2. Port B Control & Usage

		I/O MODE		LATCH MODE	
		MCR4 = 0		MCR4 = 1 (2)	
PIN NO. R65F11	PIN NO. R65F12	SIGNAL NAME	SIGNAL TYPE (1)	SIGNAL NAME	SIGNAL TYPE
38	8	PB0	I/O	PB0	INPUT
37	7	PB1	I/O	PB1	INPUT
36	6	PB2	I/O	PB2	INPUT
35	5	PB3	I/O	PB3	INPUT
34	4	PB4	I/O	PB4	INPUT
33	3	PB5	I/O	PB5	INPUT
32	2	PB6	I/O	PB6	INPUT
31	1	PB7	I/O	PB7	INPUT

(1) RESISTIVE PULL-UP, ACTIVE BUFFER PULL DOWN
(2) INPUT DATA IS STORED IN PORT B LATCH BY PA0 PULSE

TABLE 4-3. Port A Control & Usage

R65F11/R65F12 PORT[5]	PA0 I/O		PORT B LATCH MODE	
	MCR4 = 0		MCR4 = 1	
	SIGNAL		SIGNAL	
	NAME	TYPE	NAME	TYPE
PA0[2]	PA0	I/O	PORT B LATCH STROBE	INPUT[1]

	PA1-PA3 I/O	
	SIGNAL	
	NAME	TYPE
PA1[2] PA2[3] PA3[3]	PA1 PA2 PA3	I/O I/O I/O

PA4	PA4 I/O		COUNTER A I/O			
	MCR0 = 0 MCR1 = 0 SCCR7 = 0 RCVR S/R MODE = 0[4]		MCR0 = 1 MCR1 = 0 SCCR7 = 0 RCVR S/R MODE = 0[4]		SCCR7 = 0 SCCR6 = 0 MCR1 = 1	
	SIGNAL		SIGNAL		SIGNAL	
	NAME	TYPE	NAME	TYPE	NAME	TYPE
	PA4	I/O	CNTA	OUTPUT	CNTA	INPUT (1)
	SERIAL I/O SHIFT REGISTER CLOCK					
	SCCR7 = 1 SCCR5 = 1		RCVR S/R MODE = 1[4]			
	SIGNAL		SIGNAL			
	NAME	TYPE	NAME	TYPE		
	XMTR CLOCK	OUTPUT	RCVR CLOCK	INPUT (1)		

PA5	PA5 I/O		COUNTER B I/O			
	MCR3 = 0 MCR2 = 0		MCR3 = 0 MCR2 = 1		MCR3 = 1 MCR2 = X	
	SIGNAL		SIGNAL		SIGNAL	
	NAME	TYPE	NAME	TYPE	NAME	TYPE
	PA5	I/O	CNTB	OUTPUT	CNTB	INPUT (1)

PA6	PA6 I/O		SERIAL I/O XMTR OUTPUT	
	SCCR7 = 0		SCCR7 = 1	
	SIGNAL		SIGNAL	
	NAME	TYPE	NAME	TYPE
	PA6	I/O	XMTR	OUTPUT

PA7	PA7 I/O		SERIAL I/O RCVR INPUT	
	SCCR6 = 0		SCCR6 = 1	
	SIGNAL		SIGNAL	
	NAME	TYPE	NAME	TYPE
	PA7	I/O	RCVR	INPUT (1)

(1) HARDWARE BUFFER FLOAT
(2) POSITIVE EDGE DETECT
(3) NEGATIVE EDGE DETECT
(4) RCVR S/R MODE = 1 WHEN SCCR6 · SCCR5 · SCCR4 = 1
(5) APPLIES TO EITHER R65F11 OR R65F12 PORT (SEE PIN DIAGRAM)

4.5 PORT C (PC)

Port C is preprogrammed as part of the Address/Data bus. PC0-PC7 function as A0-A3, A12, R/W, A13, and $\overline{EMS}$, respectively, as shown in Table 4-4. $\overline{EMS}$ (External Memory Select) is asserted (low) whenever the internal processor accesses memory area between 0100 and 3FFF. (See Memory Map, Appendix C). The leading edge of $\overline{EMS}$ may be used to strobe the eight address lines multiplexed on Port D. See Appendix F.3 for Port C timing.

4.6 PORT D (PD)

Port D is also preprogrammed as part of the Address/Data bus. Data bits D0 through D7 are time multiplexed with address bits A4 through A11, respectively. Refer to the Memory Maps (Appendix C) for Multiplexed memory assignments. See Appendix F.3 for Port D timing.

4.7 PORT E (PE), PORT F (PF), PORT G (PG)

Ports E, F and G are available on the R65F12 only. Port E can only be used as outputs. Port F and Port G can be used for inputs or outputs and are similar to Port A and Port B in operation.

TABLE 4-4. Port C Control & Usage

R65F11/ R65F12 PORT	MULTIPLEXED MODE MCR7 = 1 MCR6 = 1	
	SIGNAL	
	NAME	TYPE (1)
PC0	A0	OUTPUT
PC1	A1	OUTPUT
PC2	A2	OUTPUT
PC3	A3	OUTPUT
PC4	A12	OUTPUT
PC5	R/W	OUTPUT
PC6	A13	OUTPUT
PC7	EMS	OUTPUT

TABLE 4-5. Port D Control & Usage

R65F11/ R65F12 PORT	MULTIPLEXED MODE MCR7 = 1 MCR6 = 1 MCR5 = 1			
	SIGNAL		SIGNAL	
	PHASE 1		PHASE 2	
	NAME	TYPE (2)	NAME	TYPE (3)
PD0	A4	OUTPUT	DATA0	I/O
PD1	A5	OUTPUT	DATA1	I/O
PD2	A6	OUTPUT	DATA2	I/O
PD3	A7	OUTPUT	DATA3	I/O
PD4	A8	OUTPUT	DATA4	I/O
PD5	A9	OUTPUT	DATA5	I/O
PD6	A10	OUTPUT	DATA6	I/O
PD7	A11	OUTPUT	DATA7	I/O

(1) ACTIVE BUFFER PULL-UP AND PULL-DOWN
(2) TRI-STATE BUFFER IS IN ACTIVE MODE
(3) TRI-STATE BUFFER IS IN ACTIVE MODE ONLY DURING THE PHASE 2 PORTION OF A WRITE CYCLE

SECTION 5
SERIAL INPUT/OUTPUT CHANNEL

The R65F11 and R65F12 Microcomputers provide a full duplex Serial I/O channel with programmable bit rates and operating modes. The serial I/O functions are controlled by the Serial Communication Control Register (SCCR). The SCCR bit assignment is shown in Figure 5-1. The serial bit rate is determined by Counter A for all modes except the Receiver Shift Register (RCVR S/R) mode for which an external shift clock must be provided. The maximum data rate using the internal clock is 62.5K bits per second (@ Ø2 = 1 MHZ). The transmitter (XMTR) and receiver (RCVR) can be independently programmed to operate in different modes and can be independently enabled or disabled.

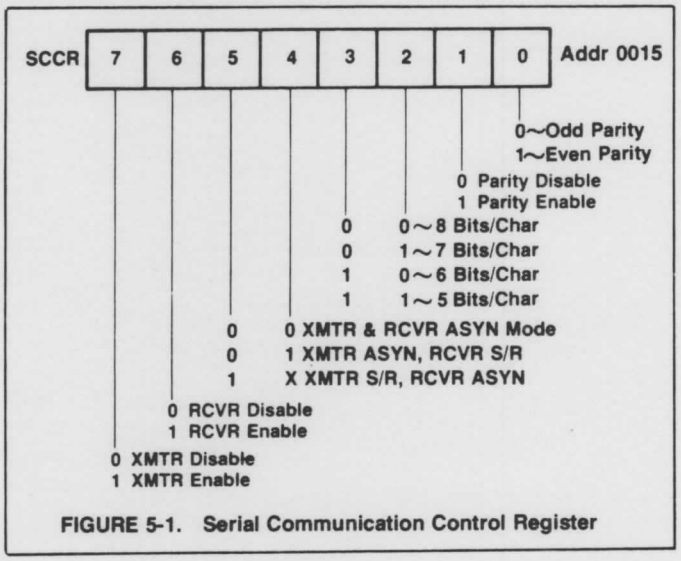


FIGURE 5-1. Serial Communication Control Register

Except for the Receiver Shift Register Mode (RCVR S/R), all XMTR and RCVR bit rates will occur at one sixteenth of the Counter A interval timer rate. Counter A is forced into an interval timer mode whenever the serial I/O is enabled in a mode requiring an internal clock.

Whenever Counter A is required as a timing source it must be loaded with the hexadecimal code that selects the data rate for the serial I/O Port. Refer to Counter A (paragraph 6.1) for a table of hexadecimal values to represent the desired data rate.

5.1 TRANSMITTER OPERATION (XTMR)

The XTMR operation and the transmitter related control/status functions are enabled by bit 7 of the Serial Communications Control Register (SCCR). The transmitter, when in the Asynchronous (ASYN) mode, automatically adds a start bit, one or two stop bits, and, when enabled, a parity bit to the transmitted data. A word of transmitted data (in asynchronous parity mode) can have 5, 6, 7, or 8 bits of data. The nine data modes are shown below. When parity is disabled, the 5, 6, 7 or 8 bits of data are terminated with two stop bits.

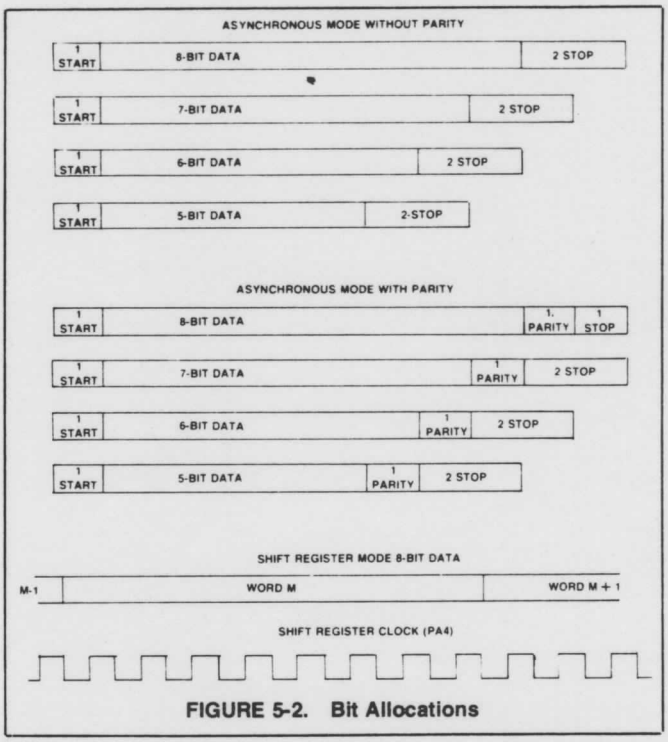


FIGURE 5-2. Bit Allocations

In the S/R mode, eight data bits are always shifted out. Bits/character and parity control bits are ignored. The serial data is shifted out via the SO output (PA6) and the shift clock is available at the CA (PA4) pin. When the transmitter underruns in the S/R mode the SO output and shift clock are held in a high state.

The XMTR Interrupt Flag bit (IFR7) is controlled by Serial Communication Status Register bits SCCR5, SCCR6 and SCCR7.

$$IFR7 = SCSR6\ (\overline{SCCR5} + SCCR7)$$

5.2 RECEIVER OPERATION (RCVR)

The receiver and its selected control and status functions are enabled when SCCR-6 is set to a "1." In the ASYN mode, data format must have a start bit, appropriate number of data bits, a parity bit (if enabled) and one stop bit. Refer to Figure 5-2 for a diagram of bit allocations. The receiver bit period is divided into 8 sub-intervals for internal synchronization. The receiver bit stream is synchronized by the start bit and a strobe signal is generated at the approximate center of each incoming bit. Refer to Figure 5-3 for ASYN Receive Data Timing. The character assembly process does not start if the start bit signal is less than one-half the bit time after a low level is detected on the Receive Data Input. Framing error, over-run, and parity error conditions or a RCVR Data Register Full will set the appropriate status bits, and any of the above conditions will cause an Interrupt Request if the Receiver Interrupt Enable bit is set to logic 1.

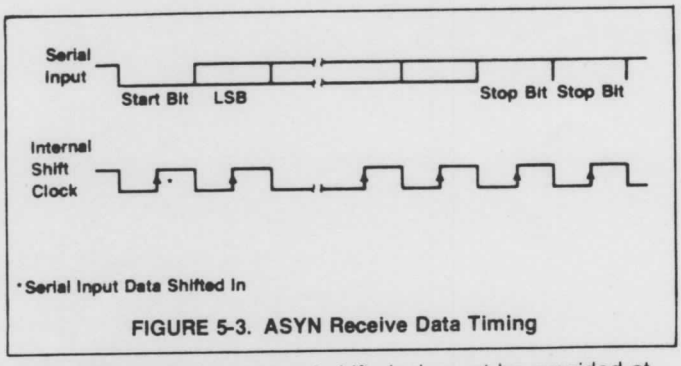


FIGURE 5-3. ASYN Receive Data Timing

In the S/R mode, an external shift clock must be provided at CA (PA4) pin along with 8 bits of serial data (LSB first) at the SI input (PA7). The maximum data rate using an external shift clock is one-eighth the internal clock rate. Refer to Figure 5-4 for S/R Mode Timing.

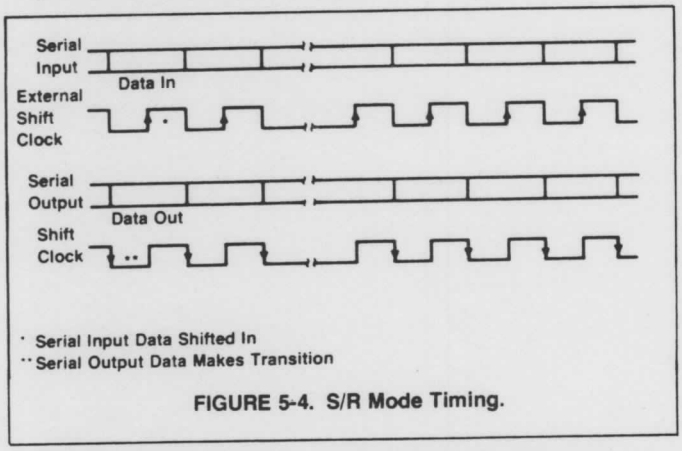


FIGURE 5-4. S/R Mode Timing.

A RCVR interrupt (IFR6) is generated whenever any of SCSR0-3 are true.

5.3 SERIAL COMMUNICATION STATUS REGISTER (SCSR)

The Serial Communication Status Register (SCSR) holds information on various communication error conditions, status of the transmitter and receiver data registers, a transmitter end-of-transmission condition, and a receiver idle line condition (Wake-Up Feature). The SCSR bit assignment is shown in Figure 5-5. Bit assignments and functions of the SCSR are as follows:

SCSR 0: *Receiver Data Register Full*—Set to a logic 1 when a character is transferred from the Receiver Shift Register to the Receiver Data Register. This bit is cleared by reading the Receiver Data Register, or by $\overline{RES}$ and is disabled if SCCR 6 = 0. The SCSR 0 bit will not be set to a logic 1 if the received data contains an error condition, however, a corresponding error bit will be set to a logic 1 instead.

SCSR 1: *Over-Run Error*—Set to a logic 1 when a new character is transferred from the Receiver Shift Register, with the last character still in the Receiver Data Register. This bit is cleared by reading the Receiver Data Register, or by $\overline{RES}$.

SCSR 2: *Parity Error*—Set to logic 1 when the RCVR is in the ASYN Mode, Parity Enable bit is set, and the received data has a parity error. This bit is cleared by reading the Receiver Data Register or by $\overline{RES}$.

SCSR 3: *Framing Error*—Set to a logic 1 when the received data contains a zero bit after the last data or parity bit in the stop bit slot. Cleared by reading the Receiver Data Register or by $\overline{RES}$. (ASYN Mode only).

SCSR 4: *Wake-Up*—Set to a logic 1 by writing a "1" in bit 4 of address: 0016. The Wake-Up bit is cleared by $\overline{RES}$ or when the receiver detects a string of ten consecutive 1's. When the Wake-Up bit is set SCSR0 through SCSR3 are inhibited.

SCSR 5: *End of Transmission*—Set to a logic 1 by writing a "1" in bit position 5 of address: 0016. The End of Transmission bit is cleared by $\overline{RES}$ or upon writing a new data word into the Transmitter Data Register. When the End-of-Transmission bit is true the Transmitter Register Empty bit is disabled until a Transmitter Under-Run occurs.

SCSR 6: *Transmitter Data Register Empty*—Set to a logic 1 when the contents of the Transmitter Data Register is transferred to the Transmitter Shift Register. Cleared upon writing new data into the Transmit Data Register. This bit is initialized to a logic 1 by $\overline{RES}$.

SCSR 7: *Transmitter Under-Run*—Set to a logic 1 when the last data bit is transmitted if the transmitter is in a S/R Mode or when the last stop bit is transmitted if the XMTR is in the ASYN Mode while the Transmitter Data Register Empty Bit is set. Cleared by a transfer of new data into the Transmitter Shift Register, or by $\overline{RES}$.

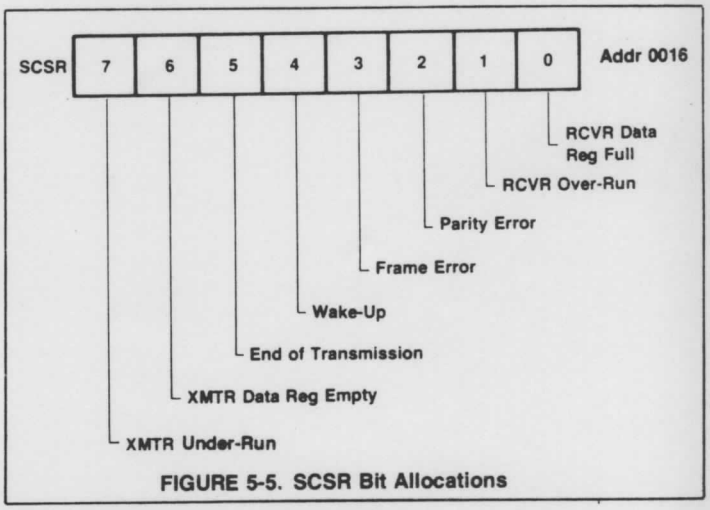


FIGURE 5-5. SCSR Bit Allocations

5.4 WAKE-UP FEATURE

In a multi-distributed microprocessor or microcomputer applications, a destination address is usually included at the beginning of the message. The Wake-Up Feature allows non-selected CPU's to ignore the remainder of the message until the beginning of the next message by setting the Wake-Up bit. As long as the Wake-Up flag is true, the Receiver Data Register Full Flag remains false. The Wake-Up bit is automatically cleared when the receiver detects a string of ten consecutive 1's which indicates an idle transmit line. When the next byte is received, the Receiver Data Register Full Flag signals the CPU to wake-up and read the received data.

SECTION 6
COUNTER/TIMERS

The R65F11 and R65F12 Microcomputers contain two 16-bit counters (Counter A and Counter B) and three 16-bit latches associated with the counters. Counter A has one 16-bit latch and Counter B has two 16-bit latches. Each counter can be independently programmed to operate in one of four modes:

Counter A	Counter B
• Pulse width measurement	• Retriggerable Interval Counter
• Pulse Generation	• Asymmetrical Pulse Generation
• Interval Timer	• Interval Timer
• Event Counter	• Event Counter

Operating modes of Counter A and Counter B are controlled by the Mode Control Register. All counting begins at the initialization value and decrements. When modes are selected requiring a counter input/output line, PA4 is automatically selected for Counter A and PA5 is automatically selected for Counter B (see Table 4.2).

6.1 COUNTER A

Counter A consists of a 16-bit counter and a 16-bit latch organized as follows: Lower Counter A (LCA), Upper Counter A (UCA), Lower Latch A (LLA), and Upper Latch A (ULA). The counter contains the count of either Ø2 clock pulses or external events, depending on the counter mode selected. The contents of Counter A may be read any time by executing a read at location 0019 for the Upper Counter A and at location 001A or location 0018 for the Lower Counter A. A read at location 0018 also clears the Counter A Underflow Flag (IFR4).

The 16-bit latch contains the counter initialization value, and can be loaded at any time by executing a write to the Upper Latch A at location 0019 and the Lower Latch A at location 0018. In either case, the contents of the accumulator are copied into the applicable latch register.

Counter A can be started at any time by writing to address: 001A. The contents of the accumulator will be copied into the Upper Latch A before the contents of the 16-bit latch are transferred to Counter A. Counter A is set to the latch value whenever Counter A underflows. When Counter A decrements from 0000 the next counter value will be the latch value, not FFFF, and the Counter A Underflow Flag (IFR 4) will be set to "1". This bit may be cleared by reading the Lower Counter A at location 0018, by writing to address location 001A, or by $\overline{RES}$.

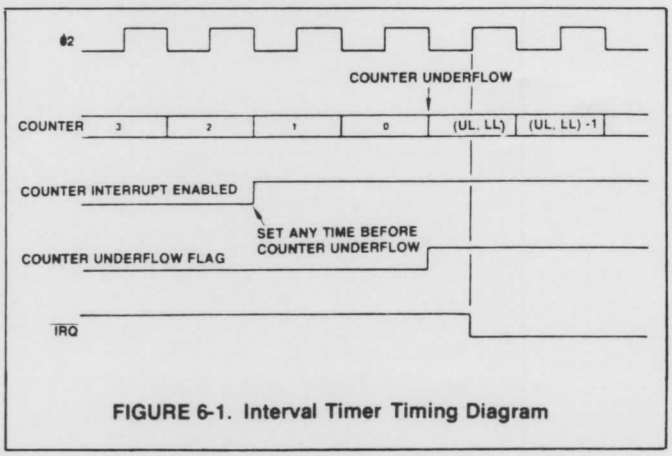


FIGURE 6-1. Interval Timer Timing Diagram

Counter A operates in any of four modes. These modes are selected by the Counter A Mode Control bits in the Control Register. See Table 6-1.

TABLE 6-1. Counter A Control Bits

MCR1 (bit 1)	MCR0 (bit 0)	Mode
0	0	Interval Timer
0	1	Pulse Generation
1	0	Event Counter
1	1	Pulse Width Measurement

The Interval Timer, Pulse Generation, and Pulse Width Measurement Modes are Ø2 clock counter modes. The Event Counter Mode counts the occurrences of an external event on the CNTR line.

The Counter is set to the Interval Timer Mode (00) when a $\overline{RES}$ signal is generated.

6.1.1 Interval Timer

In the Interval Timer mode the Counter is initialized to the Latch value by either of two conditions:-

1. When the Counter is decremented from 0000, the next Counter value is the Latch value (not FFFF).

2. When a write operation is performed to the Load Upper Latch and Transfer Latch to Counter address 001A, the Counter is loaded with the Latch value. Note that the contents of the Accumulator are loaded into the Upper Latch before the Latch value is transferred to the Counter.

The Counter value is decremented by one count at the Ø2 clock rate. The 16-bit Counter can hold from 1 to 65535 counts. The Counter Timer capacity is therefore 1μs to 65 535 ms at the 1 MHz Ø2 clock rate or 0.5 μs to 32.767 ms at the 2 MHz Ø2 clock rate. Time intervals greater than the maximum Counter value can be easily measured by counting $\overline{IRQ}$ interrupt requests in the counter $\overline{IRQ}$ interrupt routine.

When Counter A decrements from 0000, the Counter A Underflow (IFR4) is set to logic 1. If the Counter A Interrupt Enable Bit (IER4) is also set, an $\overline{IRQ}$ interrupt request will be generated. The Counter A Underflow bit in the Interrupt Flag Register can be examined in the $\overline{IRQ}$ interrupt routine to determine that the $\overline{IRQ}$ was generated by the Counter A Underflow.

While the timer is operating in the Interval Timer Mode, PA4 operates as a PA I/O bit.

A timing diagram of the Interval Timer Mode is shown in Figure 6-1.

6.1.2 Pulse Generation Mode

In the Pulse Generation mode, the CA line operates as a Counter Output. The line toggles from low to high or from high to low whenever a Counter A Underflow occurs, or a write is performed to address 001A.

The normal output waveform is a symmetrical square-wave. The CA output is initialized high when entering the mode and transitions low when writing to 001A.

Asymmetric waveforms can be generated if the value of the latch is changed after each counter underflow.

A one-shot waveform can be generated by changing from Pulse Generation to Interval Timer mode after only one occurrence of the output toggle condition.

6.1.3 Event Counter Mode

In this mode the CA is used as an Event Input line, and the Counter will decrement with each rising edge detected on this line. The maximum rate at which this edge can be detected is one-half the ϕ2 clock rate.

The Counter can count up to 65,535 occurrences before underflowing. As in the other modes, the Counter A Underflow bit (IER4) is set to logic 1 if the underflow occurs.

Figure 6.2 is a timing diagram of the Event Counter Mode.

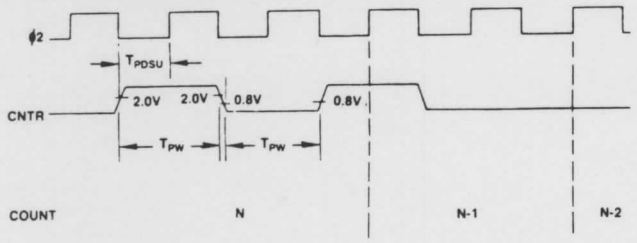


FIGURE 6-2. Event Counter Mode

6.1.4 Pulse Width Measurement Mode

This mode allows the accurate measurement of a low pulse duration on the CA line. The Counter decrements by one count at the ϕ2 clock rate as long as the CA line is held in the low state. The Counter is stopped when CA is in the high state.

The Counter A underflow flag will be set only when the count in the timer reaches zero. Upon reaching zero the timer will be loaded with the latch value and continue counting down as long as the CA pin is held low. After the counter is stopped by a high level on CA, the count will hold as long as CA remains high. Any further low levels on CA will again cause the counter to count down from its present value. The state of the CA line can be determined by testing the state of PA4.

A timing diagram for the Pulse Width Measurement Mode is shown in Figure 6-3.

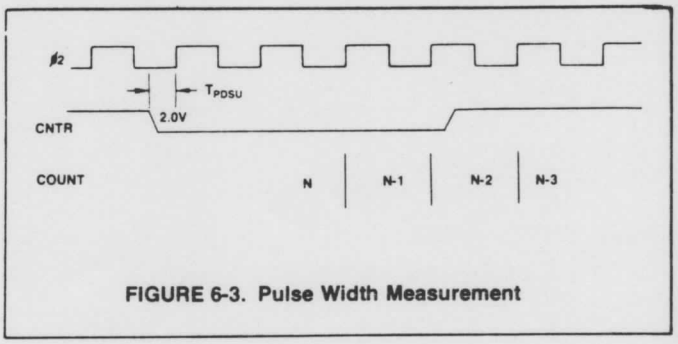


FIGURE 6-3. Pulse Width Measurement

6.1.5 Serial I/O Data Rate Generation

Counter A also provides clock timing for the Serial I/O which establishes the data rate for the Serial I/O port. When the Serial I/O is enabled, Counter A is forced to operate at the internal clock rate. Counter A is not required for the RCVR S/R mode. The Counter I/O (PA4) may also be required to support the Serial I/O (see Table 4-2).

Table 6-2 identifies the values to be loaded in Counter A for selecting standard data rates with a ϕ2 clock rate of 1 MHz and 2 MHz. Although Table 6-2 identifies only the more common data rates, any data rate from 1 to 62.5K bps can be selected by using the formula:

$$N = \frac{\phi 2}{16 \times bps} - 1$$

where

N = decimal value to be loaded into Counter A using its hexadecimal equivalent.
ϕ2 = the clock frequency (1 MHz or 2 MHz)
bps = the desired data rate.

NOTE

In Table 6-2 you will notice that the standard data rate and the actual data rate may be slightly different. Transmitter and receiver errors of 1.5% or less are acceptable. A revised clock rate is included in Table 6-2 for those baud rates which fall outside this limit.

TABLE 6-2. Counter A Values for Baud Rate Selection

STANDARD BAUD RATE	HEXADECIMAL VALUE		ACTUAL BAUD RATE AT		CLOCK RATE NEEDED TO GET STANDARD BAUD RATE	
	1 MHz	2 MHz	1 MHz	2 MHz	1 MHz	2 MHz
50	04E1	09C3	50.00	50.00	1.0000	2.0000
75	0340	0682	75.03	74.99	1.0000	2.0000
110	0237	046F	110.04	110.04	1.0000	2.0000
150	01A0	0340	149.88	150.06	1.0000	2.0000
300	00CF	01A0	300.48	299.76	1.0000	2.0000
600	0067	00CF	600.96	600.96	1.0000	2.0000
1200	0033	0067	1201.92	1201.92	1.0000	2.0000
2400	0019	0033	2403.85	2403.85	1.0000	2.0000
3600	0010	0021	3676.47	3676.47	0.9792	1.9584
4800	000C	0019	4807.69	4807.69	1.0000	2.0000
7200	0008	0010	6944.44	7352.94	1.0368	1.9584
9600	0006	000C	8928.57	9615.38	1.0752	2.0000

6.2 COUNTER B

Counter B consists of a 16-bit counter and two 16-bit latches organized as follows: Lower Counter B (LCB), Upper Counter B (UCB), Lower Latch B (LLB), Upper Latch B (ULB), Lower Latch C (LLC), and Upper Latch C (ULC). Latch C is used only in the asymmetrical pulse generation mode. The counter contains the count of either Ø2 clock pulses or external events depending on the counter mode selected. The contents of Counter B may be read any time by executing a read at location 001D for the Upper Counter B and at location 001E or 001C for the Lower Counter B. A read at location 001C also clears the Counter B Underflow Flag.

Latch B contains the counter initialization value, and can be loaded at any time by executing a write to the Upper Latch B at location 001D and the Lower Latch B at location 001C. In each case, the contents of the accumulator are copied into the applicable latch register.

Counter B can be initialized at any time by writing to address: 001E. The contents of the accumulator is copied into the Upper Latch B before the value in the 16-bit Latch B is transferred to Counter B. Counter B will also be set to the latch value and the Counter B Underflow Flag bit (IFR5) will be set to a "1" whenever Counter B underflows by decrementing from 0000.

IFR 5 may be cleared by reading the Lower Counter B at location 001C, by writing to address location 001E, or by $\overline{RES}$.

Counter B operates in the same manner as Counter A in the Interval Timer and Event Counter modes. The Pulse Width Measurement Mode is replaced by the Retriggerable Interval Timer mode and the Pulse Generation mode is replaced by the Asymmetrical Pulse Generation Mode.

6.2.1 Retriggerable Interval Timer Mode

When operating in the Retriggerable Interval Timer mode, Counter B is initialized to the latch value by writing to address 001E, by a Counter B underflow, or whenever a positive edge occurs on the CB pin (PA5). The Counter B interrupt flag will be set if the counter underflows before a positive edge occurs on the CB line. Figure 6-4 illustrates the operation.

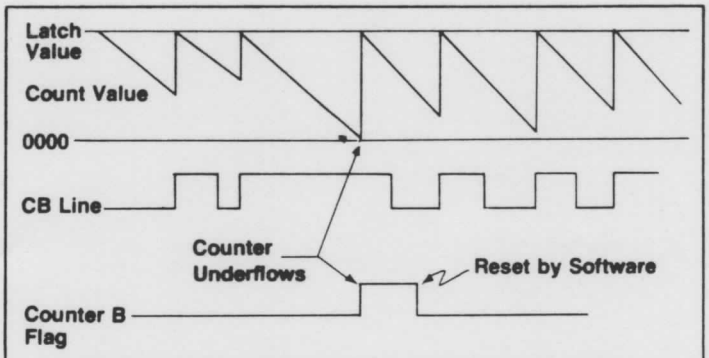


FIGURE 6-4 Counter B. Retriggerable Interval Timer Mode

6.2.2 Asymmetrical Pulse Generation Mode

Counter B has a special Asymmetrical Pulse Generation Mode whereby a pulse train with programmable pulse width and period can be generated without the processor intervention once the latch values are initialized.

In this mode, the 16-bit Latch B is initialized with a value which corresponds to the duration between pulses (referred to as D in the following descriptions). The 16-bit Latch C is initialized with a value which corresponds to the desired pulse width (referred to as P in the following descriptions). The initialization sequence for Latch B and C and the starting of a counting sequence are as follows:

1. The lower 8 bits of P are loaded into LLB by writing to address 001C, and the upper 8 bits of P are loaded into ULB and the full 16 bits are transferred to Latch C by writing to address location 001D. At this point both Latch B and Latch C contain the value of P.
2. The lower 8 bits of D are loaded into LLB by writing to address 001C, and the upper 8 bits of D are loaded into ULB by writing to address location 001E. Writing to address location 001E also causes the contents of the 16-bit Latch B to be downloaded into the Counter B and causes the CB output to go low as shown in Figure 6-5.
3. When the Counter B underflow occurs the contents of the Latch C is loaded into the Counter B, and the CB output toggles to a high level and stays high until another underflow occurs. Latch B is then down-loaded and the CB output toggles to a low level repeating the whole process.

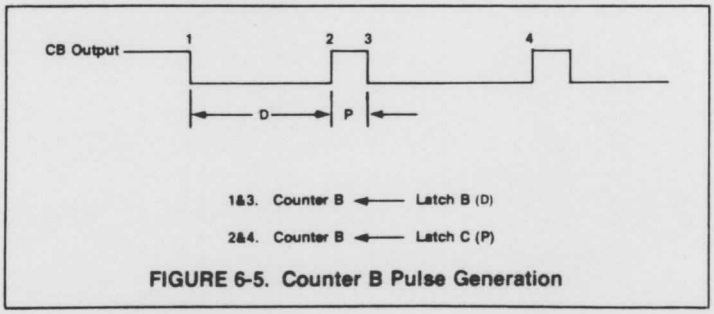


FIGURE 6-5. Counter B Pulse Generation

SECTION 7
POWER ON/INITIALIZATION CONSIDERATIONS

7.1 POWER-ON-RESET

The occurrence of $\overline{RES}$ going from low to high will cause the R65F11 or R65F12 to reset and enter the RSC-FORTH Operating System. As was described in Section 3.8, upon reset certain system variables will be initialized. See Appendix C.4 for a list of these variables names, locations and contents. The external memory map will be searched for an auto start ROM.

A bit pattern of A55A at a 1K byte page boundary indicates that an auto start program follows. The next two bytes are assumed to be a pointer to the high level RSC-FORTH word that is the entry point to that program. Auto start programs is written in assembly language, rather than RSC-FORTH, a series of indirect pointers as shown in 3-7 can be used to initiate program execution.

7.2 POWER ON TIMING

After application of V_{CC} and V_{RR} power to the R65F11 or R65F12, $\overline{RES}$ must be held low for at least eight Ø2 clock cycles after V_{CC} reaches operating range and the internal oscillator has stabilized. This stabilization time is dependent upon the input V_{CC} voltage and performance of the internal oscillator. The clock can be monitored at Ø2 (pin 3). Figure 7-1 illustrates the power turn-on waveforms. Clock stabilization time is typically 20 ms.

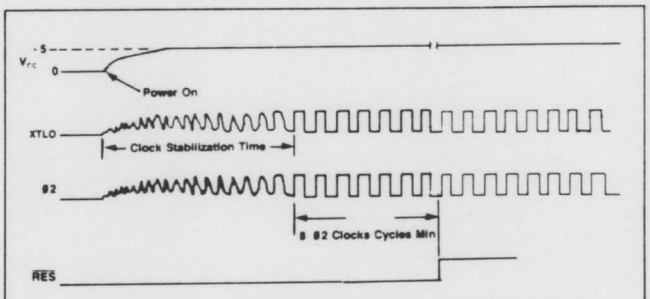


FIGURE 7-1. Power Turn-On Timing Detail

7.3 RESET ($\overline{RES}$) CONDITIONING

When $\overline{RES}$ is driven from low to high the R65F11 or R65F12 is put in a reset state. The registers and I/O ports are configured as shown in Table 7-1 when the external ROM is autostarted.

TABLE 7-1. $\overline{RES}$ Initialization of I/O Ports and Registers

	7	6	5	4	3	2	1	0
REGISTERS								
Mode Control (MCR)	1	1	1	0	0	0	0	0
Int. Enable (IER)	0	0	0	0	0	0	0	0
Int. Flag (IFR)	0	0	0	0	0	0	0	0
Ser. Com. Control (SCCR)	1	1	0	0	0	1	0	0
Ser. Com. Status (SCSR)	0	1	0	0	0	0	0	0
PORTS								
PA Latch	1	1	1	1	1	1	1	1
PB Latch	1	1	1	1	1	1	1	1

APPENDIX A
R65F11 AND R65F12 INSTRUCTION SET

This appendix contains a summary of the R6500 instruction set. For detailed information, consult the R6500 Microcomputer System Programming Manual, Document 29650 N30. The four instructions notated with a * are added instructions for the R65F11 and R65F12 which are not part of the standard 6502 instruction set.

A.1 INSTRUCTION SET IN ALPHABETIC SEQUENCE

Mnemonic	Instruction	Mnemonic	Instruction
ADC	Add Memory to Accumulator with Carry	LDA	Load Accumulator with Memory
AND	"AND" Memory with Accumulator	LDX	Load Index X with Memory
ASL	Shift Left One Bit (Memory or Accumulator)	LDY	Load Index Y with Memory
		LSR	Shift One Bit Right (Memory or Accumulator)
*BBR	Branch on Bit Reset Relative		
*BBS	Branch on Bit Set Relative		
BCC	Branch on Carry Clear	NOP	No Operation
BCS	Branch on Carry Set		
BEQ	Branch on Result Zero	ORA	"OR" Memory with Accumulator
BIT	Test Bits in Memory with Accumulator		
BMI	Branch on Result Minus	PHA	Push Accumulator on Stack
BNE	Branch on Result not Zero	PHP	Push Processor Status on Stack
BPL	Branch on Result Plus	PLA	Pull Accumulator from Stack
BRK	Force Break	PLP	Pull Processor Status from Stack
BVC	Branch on Overflow Clear		
BVS	Branch on Overflow Set	*RMB	Reset Memory Bit
		ROL	Rotate One Bit Left (Memory or Accumulator)
CLC	Clear Carry Flag		
CLD	Clear Decimal Mode	ROR	Rotate One Bit Right (Memory or Accumulator)
CLI	Clear Interrupt Disable Bit		
CLV	Clear Overflow Flag	RTI	Return from Interrupt
CMP	Compare Memory and Accumulator	RTS	Return from Subroutine
CPX	Compare Memory and Index X		
CPY	Compare Memory and Index Y	SBC	Subtract Memory from Accumulator with Borrow
DEC	Decrement Memory by One	SEC	Set Carry Flag
DEX	Decrement Index X by One	SED	Set Decimal Mode
DEY	Decrement Index Y by One	SEI	Set Interrupt Disable Status
		*SMB	Set Memory Bit
EOR	"Exclusive-Or" Memory with Accumulator	STA	Store Accumulator in Memory
		STX	Store Index X in Memory
		STY	Store Index Y in Memory
INC	Increment Memory by One		
INX	Increment Index X by One	TAX	Transfer Accumulator to Index X
INY	Increment Index Y by One	TAY	Transfer Accumulator to Index Y
		TSX	Transfer Stack Pointer to Index X
JMP	Jump to New Location	TXA	Transfer Index X to Accumulator
JSR	Jump to New Location Saving Return Address	TXS	Transfer Index X to Stack Register
		TYA	Transfer Index Y to Accumulator

A.2 R65F11 AND R65F12 INSTRUCTION SET SUMMARY TABLE

INSTRUCTIONS		R65F11 AND R65F12 ADDRESS MODES																																						PROCESSOR STATUS CODES							
		IMMEDIATE			ABSOLUTE			ZERO PAGE			ACCUM.			IMPLIED			(IND, X)			(IND), Y			Z. PAGE, X			ABS, X			ABS, Y			RELATIVE			INDIRECT			Z. PAGE, Y			BIT ADDRESSING (OP BY BIT #)								7	6	5	4	3	2	1	0
MNEMONIC	OPERATION	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	0	1	2	3	4	5	6	7	N	V	·	B	D	I	Z	C
ADC	A + M + C → A (4)(1)	69	2	2	6D	4	3	65	3	2							61	6	2	71	5	2	75	4	2	7D	4	3	79	4	3																										
AND	A M → A (1)	29	2	2	2D	4	3	25	3	2							21	6	2	31	5	2	35	4	2	3D	4	3	39	4	3																										
ASL	C ← [7 0] ← 0				0E	6	3	06	5	2	0A	2	1										16	6	2	1E	7	3																										
BBR[#(0-7)]	Branch on M_6 = 0 (5)(2)																																							0F	1F	2F	3F	4F	5F	6F	7F	·	·	·	·	·	·	·	·
BBS[#(0-7)]	Branch on M_6 = 1 (5)(2)																																							8F	9F	AF	BF	CF	DF	EF	FF	·	·	·	·	·	·	·	·
BCC	Branch on C = 0 (2)																															90	2	2														·	·	·	·	·	·	·	·
BCS	Branch on C = 1 (2)																															B0	2	2														·	·	·	·	·	·	·	·
BEQ	Branch on Z = 1 (2)																															F0	2	2														·	·	·	·	·	·	·	·
BIT	A ∧ M				2C	4	3	24	3	2																																								M	M_7	·	·	·	·	Z	·
BMI	Branch on N = 1 (2)																															30	2	2														·	·	·	·	·	·	·	·
BNE	Branch on Z = 0 (2)																															D0	2	2														·	·	·	·	·	·	·	·
BPL	Branch on N = 0 (2)																															10	2	2														·	·	·	·	·	·	·	·
BRK	Break (See Fig 1)													00	7	1																															·	·	·	·	1	·	1	·
BVC	Branch on V = 0 (2)																															50	2	2														·	·	·	·	·	·	·	·
BVS	Branch on V = 1 (2)																															70	2	2														·	·	·	·	·	·	·	·
CLC	0 → C													18	2	1																															·	·	·	·	·	·	·	0
CLD	0 → D													D8	2	1																															·	·	·	·	0	·	·	·
CLI	0 → I													58	2	1																															·	·	·	·	·	0	·	·
CLV	0 → V													B8	2	1																															·	0	·	·	·	·	·	·
CMP	A M (1)	C9	2	2	CD	4	3	C5	3	2							C1	6	2	D1	5	2	D5	4	2	DD	4	3	D9	4	3																										
CPX	X M	E0	2	2	EC	4	3	E4	3	2																																												
CPY	Y M	C0	2	2	CC	4	3	C4	3	2																																												
DEC	M 1 → M				CE	6	3	C6	5	2													D6	6	2	DE	7	3																										
DEX	X 1 → X													CA	2	1																																						
DEY	Y 1 → Y													88	2	1																																					
EOR	A ∨ M → A (1)	49	2	2	4D	4	3	45	3	2							41	6	2	51	5	2	55	4	2	5D	4	3	59	4	3																										
INC	M + 1 → M				EE	6	3	E6															F6	6	2	FE	7	3																										
INX	X + 1 → X													E8	2	1																																					
INY	Y + 1 → Y													C8	2	1																																					
JMP	Jump to New Loc				4C	3	3																													6C	5	3														
JSR	Jump Sub (See Fig 2)				20	6	3																																														
LDA	M → A (1)	A9	2	2	AD	4	3	A5	3	2							A1	6	2	B1	5	2	B5	4	2	BD	4	3	B9	4	3																										
LDX	M → X (1)	A2	2	2	AE	4	3	A6	3	2													B4	4	2	BC	4	3	BE	4	3							B6	4	2														
LDY	M → Y (1)	A0	2	2	AC	4	3	A4	3	2													56	6	2	5E	7	3																										
LSR	0 → [7 0] → C				4E	6	3	46	5	2	4A	2	1																																									
NOP	No Operation													EA	2	1																																					
ORA	A ∨ M → A (1)	09	2	2	0D	4	3	05	3	2							01	6	2	11	5	2	15	4	2	1D	4	3	19	4	3																										
PHA	A → Ms S 1 → S													48	3	1																																				
PHP	P → Ms S 1 → S													08	3	1																																				
PLA	S + 1 → S Ms → A													68	4	1																																				
PLP	S + 1 → S Ms → P													28	4	1																																				
RMB[#(0-7)]	0 → M_b (5)																																							07	17	27	37	47	57	67	77	·	·	·	·	·	·	·	·
R0L	[7 0] ← [C]				2E	6	3	26	5	2	2A	2	1										36	6	2	3E	7	3																								N	·	·	·	·	·	Z	C
R0R	[C] → [7 0]				6E	6	3	66	5	2	6A	2	1										76	6	2	7E	7	3																								N	·	·	·	·	·	Z	C
RTI	Rtrn Int (See Fig 1)													40	6	1																																				(Restored)							
RTS	Rtrn Sub (See Fig 2)													60	6	1																																						
SBC	A M C → A (1) (4)	E9	2	2	ED	4	3	E5	3	2							E1	6	2	F1	5	2	F5	4	2	FD	4	3	F9	4	3																										
SEC	1 → C													38	2	1																																						
SED	1 → D													F8	2	1																																						
SEI	1 → T													78	2	1																																						
SMB[#(0-7)]	1 → M_b (5)																																							87	97	A7	B7	C7	D7	E7	F7	·	·	·	·	·	·	·	·
STA	A → M				8D	4	3	85	3	2							81	6	2	91	6	2	95	4	2	9D	5	3	[illegible]	5	3																										
STX	X → M				8E	4	3	86	3	2																													96	4	2														
STY	Y → M				8C	4	3	84	3	2													94	4	2																													
TAX	A → X													AA	2	1																																						
TAY	A → Y													A8	2	1																																					
TSX	S → X													BA	2	1																																					
TXA	X → A													8A	2	1																																					
TXS	X → S													9A	2	1																																					
TYA	Y → A													98	2	1																																					

NOTES

1. Add 1 to N if page boundary is crossed
2. Add 1 to N if branch occurs to same page
   Add 2 to N if branch occurs to different page
3. Carry not = Borrow
4. If in decimal mode Z flag is invalid
   accumulator must be checked on zero result.
5. Effects 8-bit data field of the specified zero page address.

LEGEND

X	= Index X	M_6	= Memory Bit 6
Y	= Index Y	+	= Add
A	= Accumulator	-	= Subtract
M	= Memory per effective address	∧	= And
M_s	= Memory per stack pointer	∨	= Or
M_b	= Selecter zero page memory bit	∀	= Exclusive Or
M_7	= Memory Bit 7	n	= Number of cycles
		#	= Number of Bytes

A.3 INSTRUCTION CODE MATRIX

MSD \ LSD	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	MSD
0	BRK Implied 1 7	ORA (IND, X) 2 6				ORA ZP 2 3	ASL ZP 2 5	RMB0 ZP 2 5	PHP Implied 1 3	ORA IMM 2 2	ASL Accum 1 2			ORA ABS 3 4	ASL ABS 3 6	BBR0 ZP 3 5**	0
1	BPL Relative 2 2**	ORA (IND), Y 2 5*				ORA ZP, X 2 4	ASL ZP, X 2 6	RMB1 ZP 2 5	CLC Implied 1 2	ORA ABS, Y 3 4*				ORA ABS, X 3 4*	ASL ABS, X 3 7	BBR1 ZP 3 5**	1
2	JSR Absolute 3 6	AND (IND, X) 2 6			BIT ZP 2 3	AND ZP 2 3	ROL ZP 2 5	RMB2 ZP 2 5	PLP Implied 1 4	AND IMM 2 2	ROL Accum 1 2		BIT ABS 3 4	AND ABS 3 4	ROL ABS 3 6	BBR2 ZP 3 5**	2
3	BMI Relative 2 2**	AND (IND, Y) 2 5*				AND ZP, X 2 4	ROL ZP, X 2 6	RMB3 ZP 2 5	SEC Implied 1 2	AND ABS, Y 3 4*				AND ABS, X 3 4*	ROL ABS, X 3 7	BBR3 ZP 3 5**	3
4	RTI Implied 1 6	EOR (IND, X) 2 6				EOR ZP 2 3	LSR ZP 2 5	RMB4 ZP 2 5	PHA Implied 1 3	EOR IMM 2 2	LSR Accum 1 2		JMP ABS 3 3	EOR ABS 3 4	LSR ABS 3 6	BBR4 ZP 3 5**	4
5	BVC Relative 2 2**	EOR (IND), Y 2 5*				EOR ZP, X 2 4	LSR ZP, X 2 6	RMB5 ZP 2 5	CLI Implied 1 2	EOR ABS, Y 3 4*				EOR ABS, X 3 4*	LSR ABS, X 3 7	BBR5 ZP 3 5**	5
6	RTS Implied 1 6	ADC (IND, X) 2 6				ADC ZP 2 3	ROR ZP 2 5	RMB6 ZP 2 5	PLA Implied 1 4	ADC IMM 2 2	ROR Accum 1 2		JMP Indirect 3 5	ADC ABS 3 4	ROR ABS 3 6	BBR6 ZP 3 5**	6
7	BVS Relative 2 2**	ADC (IND, Y) 2 5*				ADC ZP, X 2 4	ROR ZP, X 2 6	RMB7 ZP 2 5	SEI Implied 1 2	ADC ABS, Y 3 4*				ADC ABS, X 3 4*	ROR ABS, X 3 7	BBR7 ZP 3 5**	7
8		STA (IND, X) 2 6			STY ZP 2 3	STA ZP 2 3	STX ZP 2 3	SMB0 ZP 2 5	DEY Implied 1 2		TXA Implied 1 2		STY ABS 3 4	STA ABS 3 4	STX ABS 3 4	BBS0 ZP 3 5**	8
9	BCC Relative 2 2**	STA (IND, Y) 2 6			STY ZP, X 2 4	STA ZP, X 2 4	STX ZP, Y 2 4	SMB1 ZP 2 5	TYA Implied 1 2	STA ABS, Y 3 5	TXS Implied 1 2			STA ABS, X 3 5		BBS1 ZP 3 5**	9
A	LDY IMM 2 2	LDA (IND, X) 2 6	LDX IMM 2 2		LDY ZP 2 3	LDA ZP 2 3	LDX ZP 2 3	SMB2 ZP 2 5	TAY Implied 1 2	LDA IMM 2 2	TAX Implied 1 2		LDY ABS 3 4	LDA ABS 3 4	LDX ABS 3 4	BBS2 ZP 3 5**	A
B	BCS Relative 2 2**	LDA (IND), Y 2 5*			LDY ZP, X 2 4	LDA ZP, X 2 4	LDX ZP, Y 2 4	SMB3 ZP 2 5	CLV Implied 1 2	LDA ABS, Y 3 4*	TSX Implied 1 2		LDY ABS, X 3 4*	LDA ABS, X 3 4*	LDX ABS, Y 3 4*	BBS3 ZP 3 5**	B
C	CPY IMM 2 2	CMP (IND, X) 2 6			CPY ZP 2 3	CMP ZP 2 3	DEC ZP 2 5	SMB4 ZP 2 5	INY Implied 1 2	CMP IMM 2 2	DEX Implied 1 2		CPY ABS 3 4	CMP ABS 3 4	DEC ABS 3 6	BBS4 ZP 3 5**	C
D	BNE Relative 2 2**	CMP (IND), Y 2 5*				CMP ZP, X 2 4	DEC ZP, X 2 6	SMB5 ZP 2 5	CLD Implied 1 2	CMP ABS, Y 3 4*				CMP ABS, X 3 4*	DEC ABS, X 3 7	BBS5 ZP 3 5**	D
E	CPX IMM 2 2	SBC (IND, X) 2 6			CPX ZP 2 3	SBC ZP 2 3	INC ZP 2 5	SMB6 ZP 2 5	INX Implied 1 2	SBC IMM 2 2	NOP Implied 1 2		CPX ABS 3 4	SBC ABS 3 4	INC ABS 3 6	BBS6 ZP 3 5**	E
F	BEQ Relative 2 2**	SBC (IND), Y 2 5*				SBC ZP, X 2 4	INC ZP, X 2 6	SMB7 ZP 2 5	SED Implied 1 2	SBC ABS, Y 3 4*				SBC ABS, X 3 4*	INC ABS, X 3 7	BBS7 ZP 3 5**	F

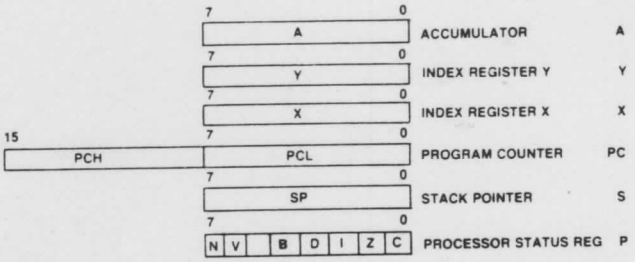
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

0

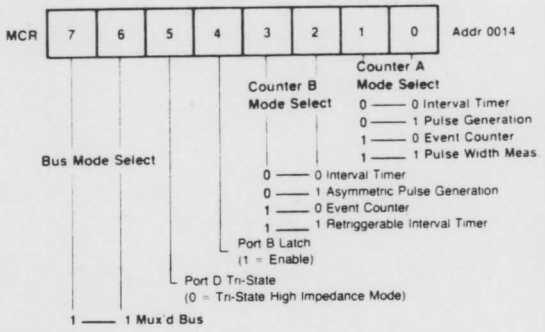
0 | BRK
Implied
1 7 | —OP Code
—Addressing Mode
—Instruction Bytes; Machine Cycles

*Add 1 to N if page boundary is crossed.
**Add 1 to N if branch occurs to same page;
add 2 to N if branch occurs to different page.

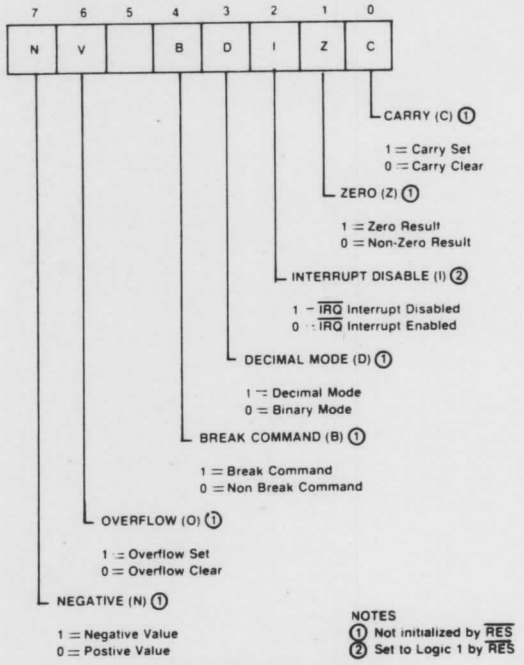
APPENDIX B
KEY REGISTER SUMMARY



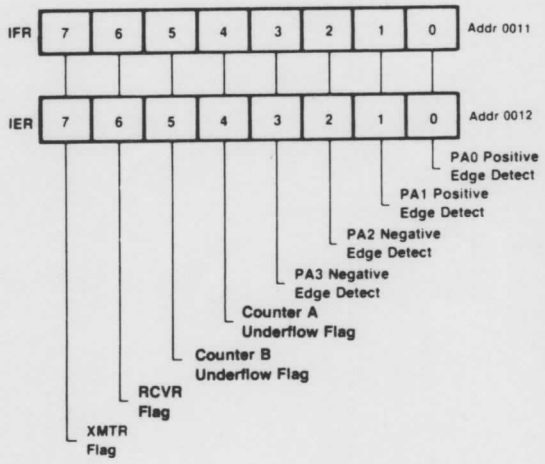
CPU Registers



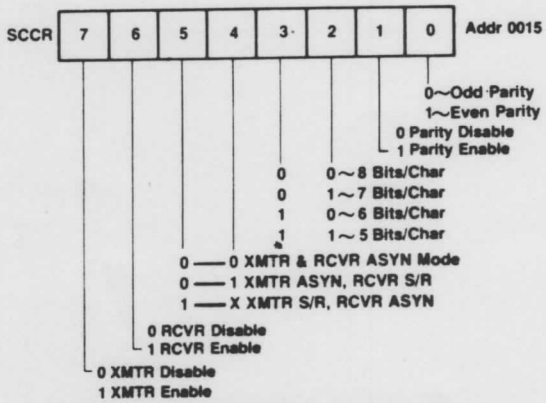
Mode Control Register



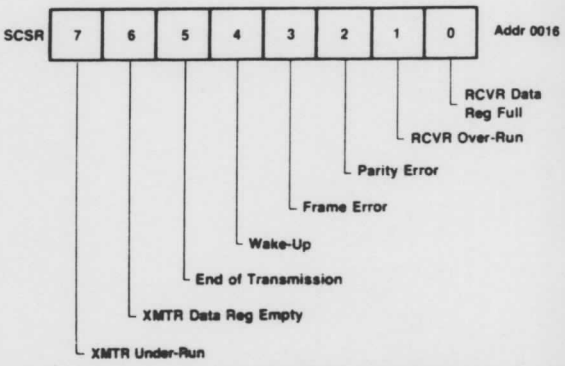
Processor Status Register



Interrupt Enable and Flag Registers



Serial Communications Control Register



Serial Communications Status Register

APPENDIX C
ADDRESS ASSIGNMENTS/MEMORY MAPS/PIN FUNCTIONS

C.1 I/O AND INTERNAL REGISTER ADDRESSES

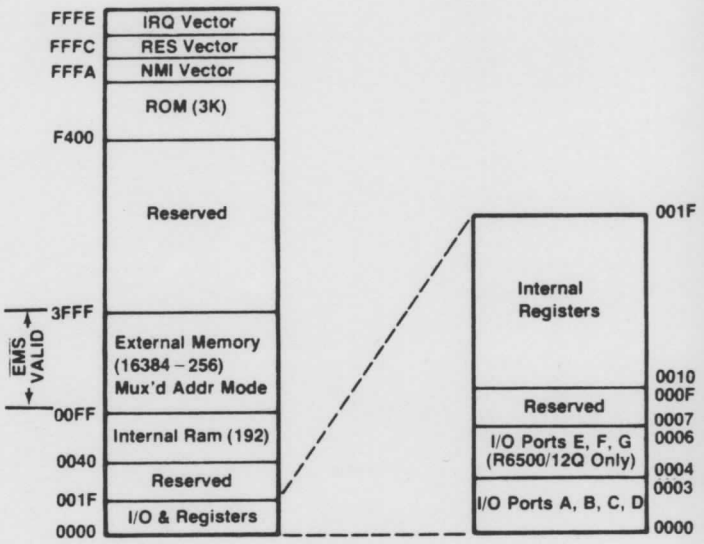
ADDRESS (HEX)	READ	WRITE
001F	— —	— —
1E	Lower Counter B	Upper Latch B, Cntr B←Latch B, CLR Flag
1D	Upper Counter B	Upper Latch B, Latch C←Latch B
1C	Lower Counter B, CLR Flag	Lower Latch B.
1B	— —	— —
1A	Lower Counter A	Upper Latch A, Cntr A←Latch A, CLR Flag
19	Upper Counter A	Upper Latch A
18	Lower Counter A, CLR Flag	Lower Latch A
17	Serial Receiver Data Register	Serial Transmitter Data Register
16	Serial Comm. Status Register	Serial Comm. Status Reg. Bits 4 & 5 only
15	Serial Comm. Control Register	Serial Comm. Control Register
14	Mode Control Register	Mode Control Register
13	— —	— —
12	Interrupt Enable Register	Interrupt Enable Register
11	Interrupt Flag Register	— —
10	Read FF	Clear Int Flag (Bits 0-3 only, Write 0's only)
0F	— —	— —
0E	— —	— —
0D	— —	— —
0C	— —	— —
0B	— —	— —
0A	— —	— —
09	— —	— —
08	— —	— —
07	— —	— —
06	Port G*	Port G*
05	Port F*	Port F*
04	Port E*	Port E*
03	— —	— —
02	— —	— —
01	Port B	Port B
0000	Port A	Port A

NOTE: *R65F12 Only

C.2 MULTIPLE FUNCTION PIN ASSIGNMENTS— PORT C AND PORT D

PIN NUMBER		I/O PORT FUNCTION REPLACED	MULTIPLEXED PORT FUNCTION
R65F11	R65F12		
4	25	PC0	A0
5	26	PC1	A1
6	27	PC2	A2
7	28	PC3	A3
8	29	PC4	A12
9	30	PC5	R/W
10	31	PC6	A13
11	32	PC7	EMS
19	40	PD0	A4/D0
18	39	PD1	A5/D1
17	38	PD2	A6/D2
16	37	PD3	A7/D3
15	36	PD4	A8/D4
14	35	PD5	A9/D5
13	34	PD6	A10/D6
12	33	PD7	A11/D7

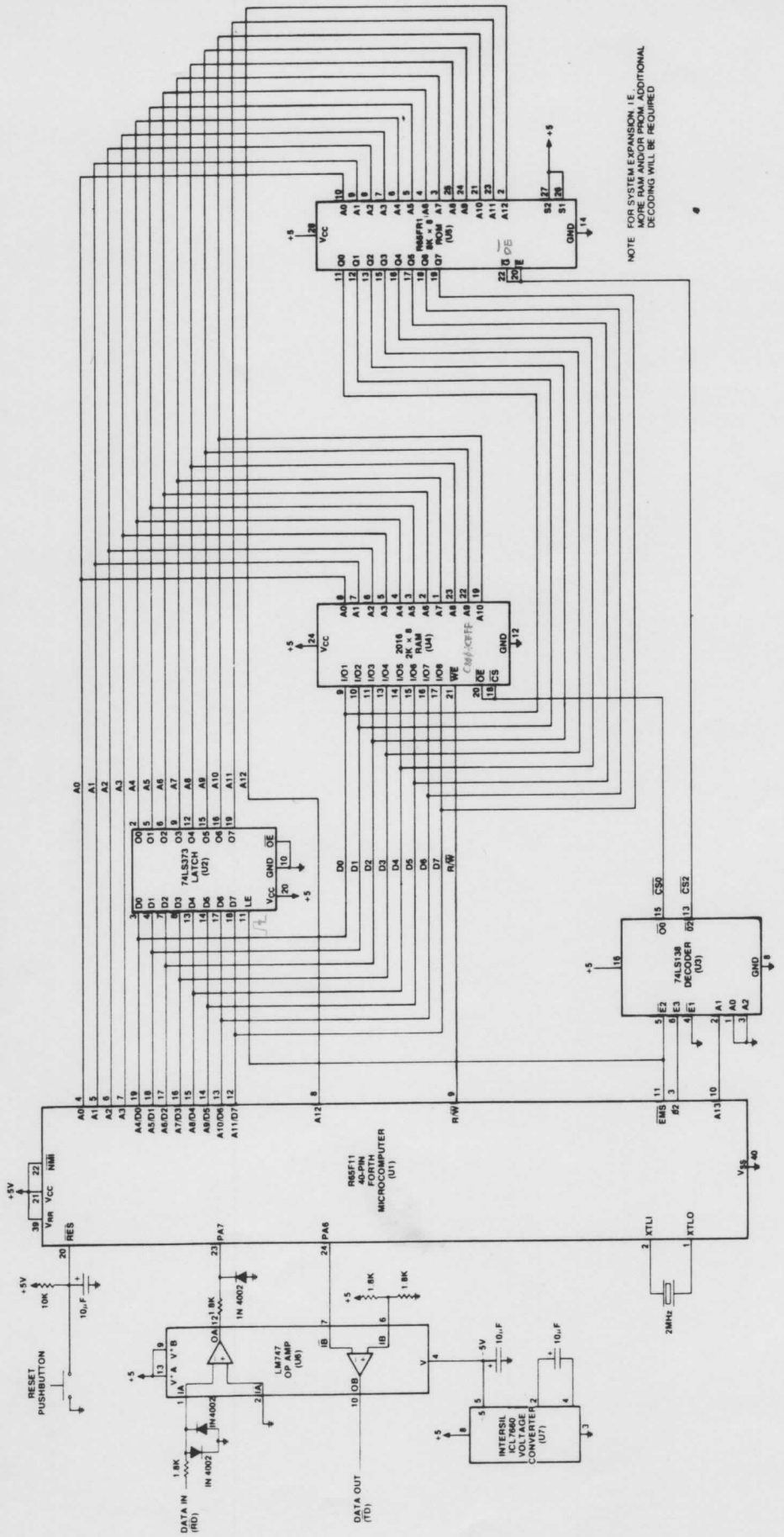
C.3 MULTIPLEXED MODE MEMORY MAP



C.4 SYSTEM VARIABLES IN RAM

ADDRESS	NAME	COLD START VALUE	WARM START VALUE
0040	IRQVEC	(COLD)	—
0042	NMIVEC	(COLD)	—
0044	UKEY	(INK)	(INK)
0046	UEMIT	(OUT)	(OUT)
0048	UP	0300	0300
004A	INTFLG	00	00
004B	(W-1)	6C	6C
004C	W	—	—
004E	IP	—	—
0050	(N-1)	—	—
0051	N	—	—
0059	XSAVE	—	—
005B	INTVEC	(COLD)	—
005D	TOS	—	—
0300	TIB	0380	0380
0302	R0	00FF	00FF
0304	S0	00C2	00C2
0306	UC/L	0050	—
0308	UPAD	037E	—
030A	UR/W	(DISK)	—
030C	BASE	0010	—
030E	CLD/WRM	—	—
0310	IN	—	—
0312	DPL	—	—
0314	HLD	—	—
0316	DISKNO	—	—
0318	CURCYL	—	—
031C	B/SIDE	—	—

APPENDIX D
TYPICAL MINIMUM HOOKUP



APPENDIX E
ELECTRICAL SPECIFICATIONS

Maximum Ratings

RATING	SYMBOL	VALUE	UNIT
Supply Voltage	V_{CC} & V_{RR}	−0.3 to +7.0	Vdc
Input Voltage	V_{in}	−0.3 to +7.0	Vdc
Operating Temperature Range, Commercial	T	0 to +70	°C
Storage Temperature Range	T_{stg}	−55 to +150	°C

This device contains circuitry to protect the inputs against damage due to high static voltages, however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum rated voltages to this circuit.

DC Characteristics (V_{CC} = 5V ±5% V_{SS} = 0; TA = 0 to 70°C)

CHARACTERISTIC	SYMBOL	MIN	TYP	MAX	UNIT
Power Dissipation (Outputs High) Commercial @ 25°C	P_D	—		− 1200	mW
RAM Standby Voltage (Retention Mode)	V_{RR}	3.0	—	V_{CC}	Vdc
RAM Standby Current (Retention Mode) Commercial @ 25°C	I_{RR}	— —	4 —	— —	mAdc
Input High Voltage (Except XTLI)	V_{IH}	+2.0	—	V_{CC}	Vdc
Input High Voltage (XTLI)	V_{IH}	+4.0	—	V_{CC}	Vdc
Input Low Voltage	V_{IL}	−0.3	—	+0.8	Vdc
Input Leakage Current ($\overline{RES}$, $\overline{NMI}$) V_{in} = 0 to 5.0 Vdc	I_{IN}			±10.0	μAdc
Input Low Current PA, PB, PC, PF*, AND PG* (V_{IL} = 0.4 Vdc)	I_{IL}	—	−1.0	−1.6	mAdc
Output High Voltage (Except XTLO) (I_{Load} = .100 μAdc)	V_{OH}	+2.4	—	V_{CC}	Vdc
Output Low Voltage (I_{Load} = 1.6 mAdc)	V_{OL}	—	—	+0.4	Vdc
Darlington Current Drive, PE* (V_O = 1.5 Vdc)	I_{OH}	−1.0	—	—	mAdc
		—	—		
Input Capacitance (V_{in} −0, T_A = 25°C, f = 1.0 MHz) PA, PB, PC, PD, PF*, and PG* XTLI, XTLO	C_{in}	 — —	 — —	 10 50	pF
I/O Port Pull-Up Resistance PA0-PA7, PB0-PB7, PC0-PC7, PF0-PF7 & PG0-PG7	R_L	3.0	6.0	11.5	KΩ
Output Leakage Current Tri-State I/O's while in High Impedance State	$I_{\phi UT}$	—	—	±10	μAdc
Output Capacitance Tri-State I/O's while in High Impedance State V_{IN} = 0V, TA = 25°C, f = 1.0 MHz	$C_{\phi UT}$	—	—	10	pF

NOTE: Negative sign indicates outward current flow, positive indicates inward flow.

*R65F12 only

APPENDIX F
TIMING REQUIREMENTS AND CHARACTERISTICS

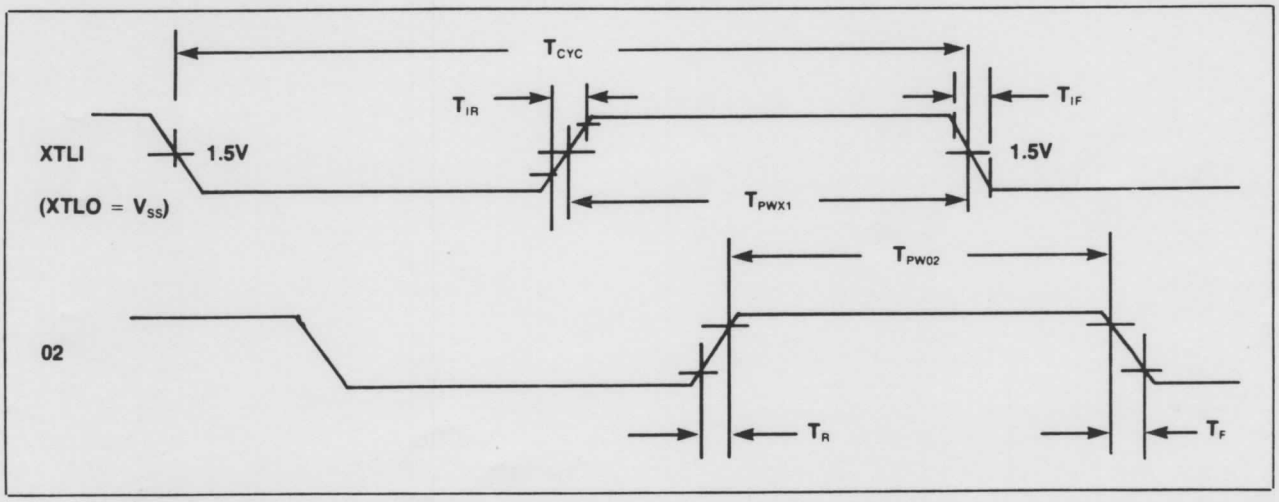
F.1 GENERAL NOTES

1. V_{CC} = 5V ± 5%, 0°C ≤ TA ≤ 70°C
2. A valid V_{CC} – RES sequence is required before proper operation is achieved.
3. All timing reference levels are 0.8V and 2.0V, unless otherwise specified.
4. All time units are nanoseconds, unless otherwise specified.
5. All capacitive loading is 130pf maximum, except as noted below:

   PA, PB, PE, PF, PG — 50pf maximum

F.2 CLOCK TIMING

SYMBOL	PARAMETER	1 MHz		2 MHz	
		MIN	MAX	MIN	MAX
T_{CYC}	Cycle Time	1000	10 μs	500	10 μs
T_{PWX1}	XTLI Input Clock Pulse Width XTLO = VSS	500 ± 25	—	250 ± 10	—
T_{PW02}	Output Clock Pulse Width at Minimum T_{CYC}	T_{PWX1}	T_{PWX1} ± 25	T_{PWX1}	T_{PWX1} ± 20
T_R, T_F	Output Clock Rise, Fall Time	—	25	—	15
T_{IR}, T_{IF}	Input Clock Rise, Fall Time	—	10	—	10



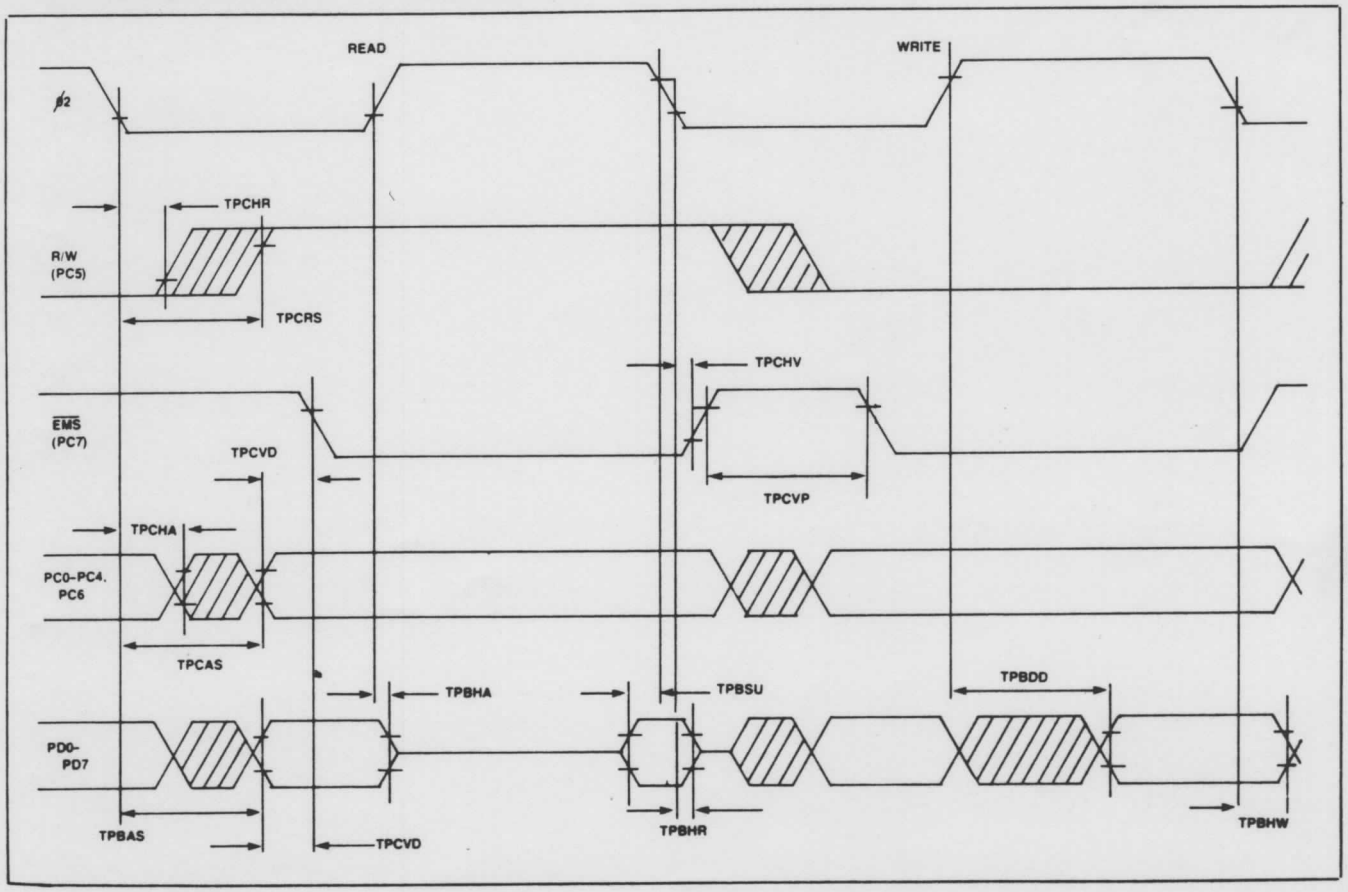
F.3 MULTIPLEXED MODE TIMING—PC AND PD

(MCR 5 = 1, MCR 6 = 1, MCR 7 = 1)

SYMBOL	PARAMETER	1 MHz		2 MHz	
		MIN	MAX	MIN	MAX
T_{PCRS}	(PC5) R/W Setup Time	—	225	—	140
T_{PCAS}	(PC0-PC4, PC6) Address Setup Time	—	225	—	140
T_{PBAS}	(PD) Address Setup Time	—	225	—	140
T_{PBSU}	(PD) Data Setup Time	50	—	35	—
T_{PBHR}	(PD) Data Read Hold Time	10	—	10	—
T_{PBHW}	(PD) Data Write Hold Time	30	—	30	—
T_{PBDD}	(PD) Data Output Delay	—	175	—	150
T_{PCHA}	(PC0-PC4, PC6) Address Hold Time	30	—	30	—
T_{PBHA}	(PD) Address Hold Time	0	100	0	80
T_{PCHR}	(PC5) R/W Hold Time	30	—	30	—
T_{PCHV}	(PC7) EMS Hold Time	10	—	10	—
T_{PCVD} (1)	(PC7) Address to EMS Delay Time	30	150	30	90
T_{PCVP}	(PC7) EMS Stabilization Time	30	—	30	—

NOTE 1: Values assume PC0-PC4, PC6 and PC7 have the same capacitive load.

F.3.1 Multiplex Mode Timing Diagram

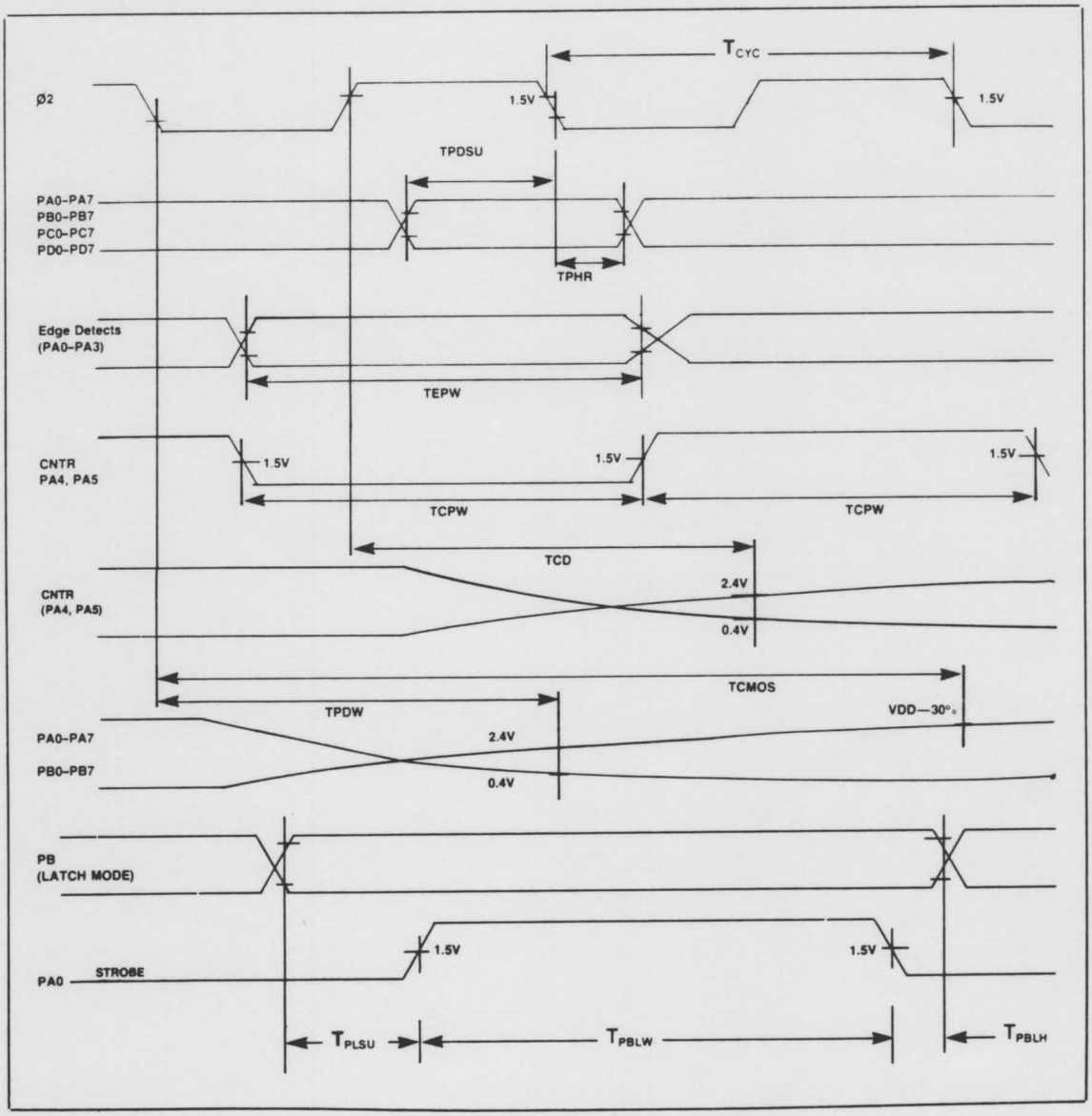


F.4 I/O, EDGE DETECT, COUNTERS, AND SERIAL I/O TIMING

SYMBOL	PARAMETER	1 MHz		2 MHz	
		MIN	MAX	MIN	MAX
	Internal Write to Peripheral Data Valid				
T_{PDW}(1)	PA, PB TTL	—	500	—	500
T_{CMOS}(1)	PA, PB CMOS	—	1000	—	1000
	Peripheral Data Setup Time				
T_{PDSU}	PA, PB	200	—	200	—
	Peripheral Data Hold Time				
T_{PHR}	PA, PB	75	—	75	—
T_{EPW}	PA0-PA3 Edge Detect Pulse Width	T_{CYC}	—	T_{CYC}	—
	Counters A and B				
T_{CPW}	PA4, PA5 Input Pulse Width	T_{CYC}	—	T_{CYC}	—
T_{CD}(1)	PA4, PA5 Output Delay	—	500	—	500
	Port B Latch Mode				
T_{PBLW}	PA0 Strobe Pulse Width	T_{CYC}	—	T_{CYC}	—
T_{PLSU}	PB Data Setup Time	175	—	150	—
T_{PBLH}	PB Data Hold Time	30	—	30	—
	Serial I/O				
T_{PDW}(1)	PA6 XMTR TTL	—	500	—	500
T_{CMOS}(1)	PA6 XMTR CMOS	—	1000	—	1000
T_{CPW}	PA4 RCVR S/R Clock Width	4 T_{CYC}	—	4 T_{CYC}	—
T_{PDW}(1)	PA4 XMTR Clock—S/R Mode (TTL)	—	500	—	500
T_{CMOS}(1)	PA4 XMTR Clock—S/R Mode (CMOS)	—	1000	—	1000

NOTE 1: Maximum Load Capacitance: 50pF Passive Pull-Up Required.

F.4.1 I/O Edge Detect, Counter, and Serial I/O Timing



APPENDIX G
INCLUDED FORTH FUNCTIONS IN ROM

BANKEXECUTE	BANKEEC!	BANKC@	BANKC!
EEC!		.R	D.
?	#S	#	SIGN
D.R	<#	SPACES	SEEK
#>	DWRITE	DREAD	SELECT
INIT	M/MOD	*/	*/MOD
DISK	/	/MOD	*
MOD	M*	MAX	MIN
M/	ABS	D+-	+-
DABS	COLD	(NUMBER)	HOLD
S->D	ERASE	FILL	QUERY
BLANKS	(.")	-TRAILING	TYPE
EXPECT	DECIMAL	HEX	-DUP
COUNT	PICK	ROT	>
SPACE	U<	=	-
<	1-	2+	1+
2-	C/L	HLD	DPL
PAD	CLD/WRM	BASE	UR/W
IN	UC/L	R0	S0
UPAD	BL	4	3
TIB	1	0	C!
2	C@	@	TOGGLE
!	BOUNDS	2DUP	DUP
+!	2DROP	DROP	OVER
SWAP	NEGATE	D+	+
DNEGATE	0=	R	R>
0<	LEAVE	;S	RP@
>R	SP!	SP@	XOR
RP!	AND	U/	U*
OR	CR	?TERMINAL	KEY
CMOVE	ENCLOSE	(FIND)	DIGIT
EMIT	(+LOOP)	(LOOP)	0BRANCH
(DO)	EXECUTE	CLIT	LIT
BRANCH			

ELECTRONIC DEVICES DIVISION REGIONAL ROCKWELL SALES OFFICES

HOME OFFICE
Electronic Devices Division
Rockwell International
4311 Jamboree Road
Newport Beach, California 92660

Mailing Address:
P.O. Box C
Newport Beach, California 92660
Mail Code 501-300
Tel: 714-833-4700
TWX: 910 591-1698

UNITED STATES
Electronic Devices Division
Rockwell International
1842 Reynolds
Irvine, California 92714
(714) 833-4655
ESL 62108710
TWX: 910 595-2518

Electronic Devices Division
Rockwell International
921 Bowser Road
Richardson, Texas 75080
(214) 996-6500
Telex: 73-307

Electronic Devices Division
Rockwell International
10700 West Higgins Rd., Suite 102
Rosemont, Illinois 60018
(312) 297-8862
TWX: 910 233-0179 (RI MED ROSM)

Electronic Devices Division
Rockwell International
5001B Greentree
Executive Campus, Rt. 73
Marlton, New Jersey 08053
(609) 596-0090
TWX: 710 940-1377

FAR EAST
Electronic Devices Division
Rockwell International Overseas Corp.
Itohpia Hirakawa-cho Bldg.
7-6, 2-chome, Hirakawa-cho
Chiyoda-ku, Tokyo 102, Japan
(03) 265-8806
Telex: J22198

EUROPE
Electronic Devices Division
Rockwell International GmbH
Fraunhoferstrasse 11
D-8033 Munchen-Martinsried
West Germany
(089) 857-6016
Telex: 0521/2650 rimd d

Electronic Devices Division
Rockwell International
Heathrow House, Bath Rd.
Cranford, Hounslow,
Middlesex, England
(01) 759-9911
Telex: 851-25463

Electronic Devices
Rockwell Collins
Via Boccaccio, 23
20123 Milano, Italy
498 74 79
Telex: 202/82

YOUR LOCAL REPRESENTATIVE

10/82